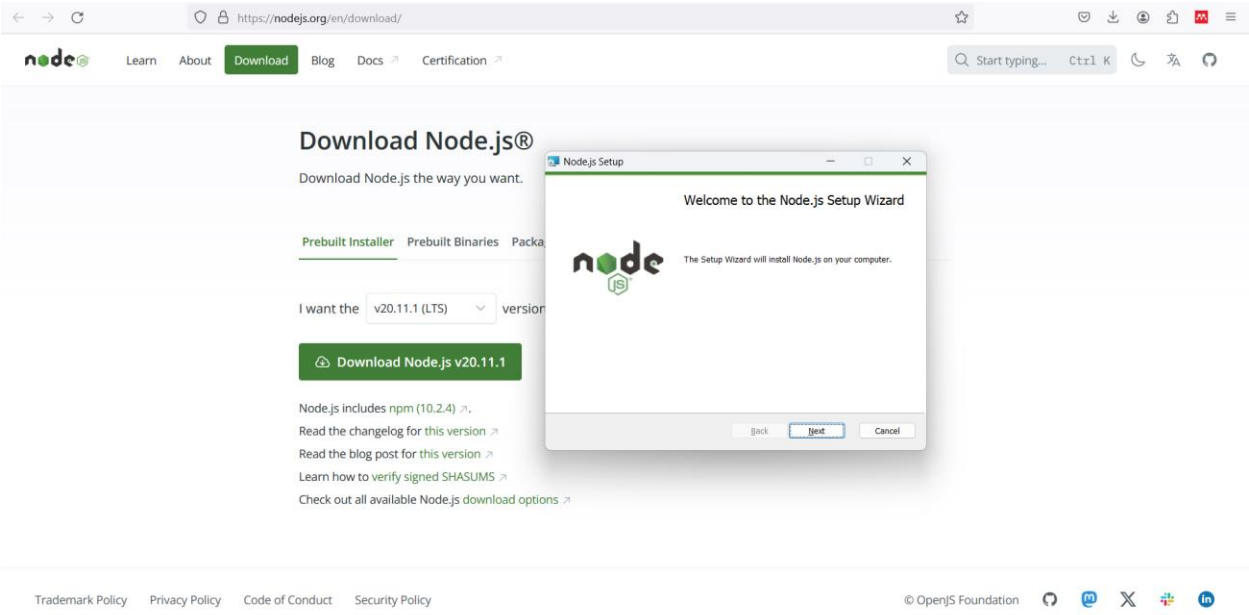
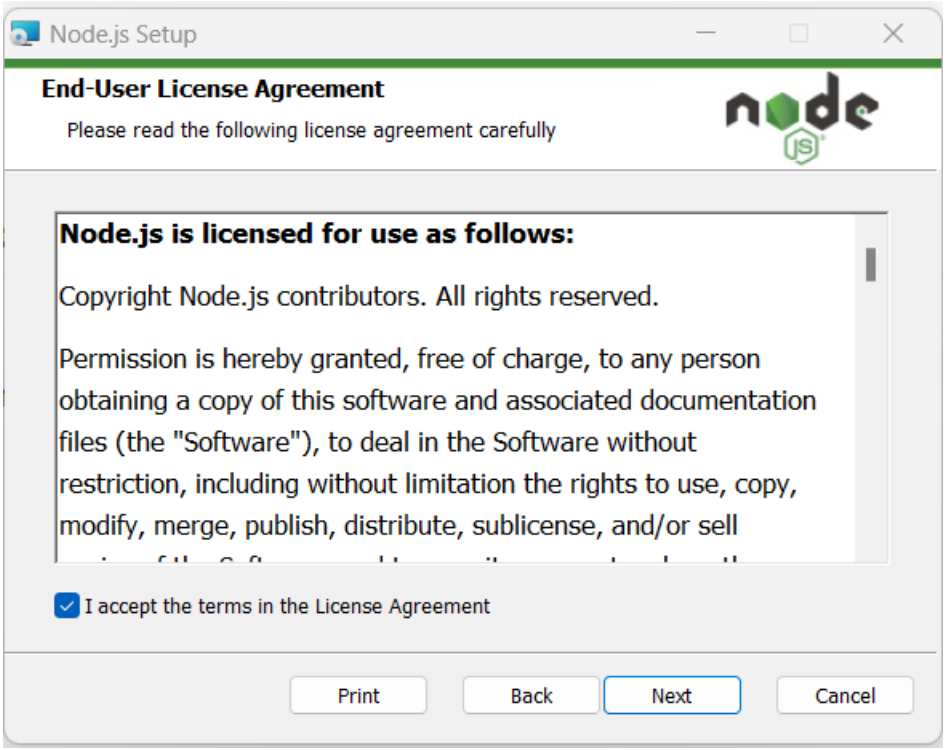
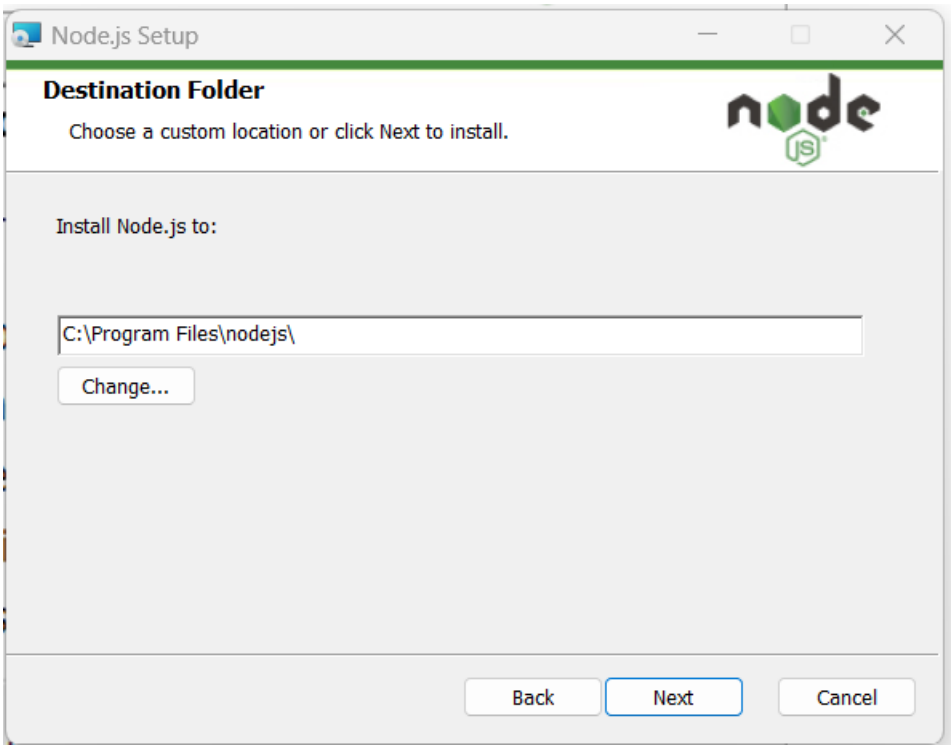
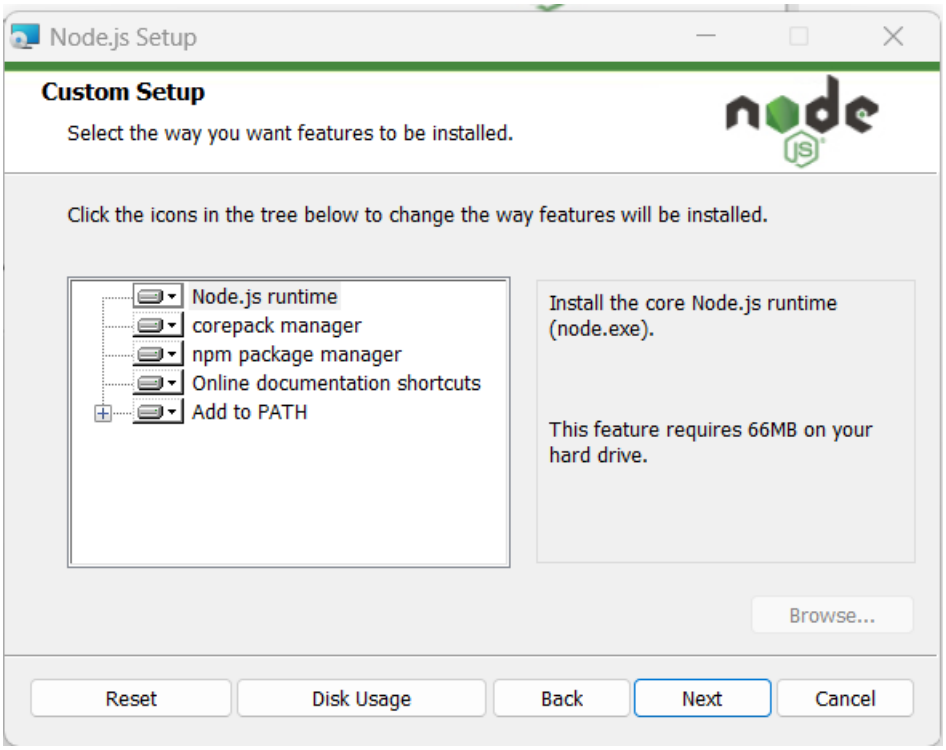
Contents

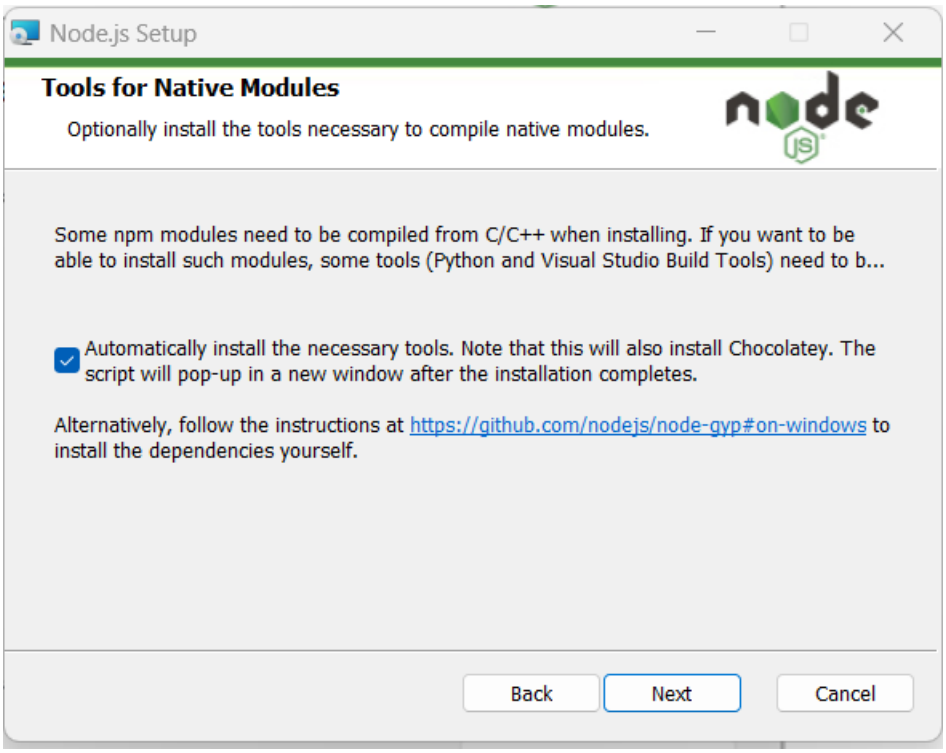
INSTALLATION

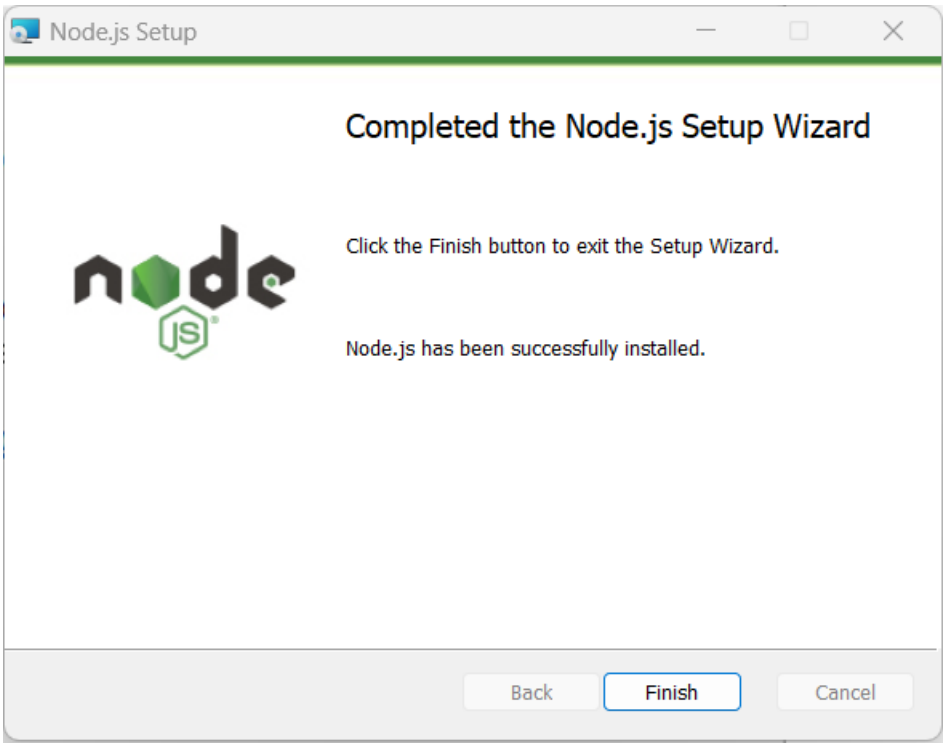





Node.js Setup
Destination Folder
Choose a custom location or click Next to install.
Install Node.js to:
C:\Program Files\nodejs\
Change...
Back
Next
Cancel


Node.js Setup
Custom Setup
Select the way you want features to be installed.
Click the icons in the tree below to change the way features will be installed.
Node.js runtime
corepack manager
npm package manager
Online documentation shortcuts
Add to PATH
Install the core Node.js runtime (node.exe).
This feature requires 66MB on your hard drive.
Browse...
Reset
Disk Usage
Back
Next
Cancel

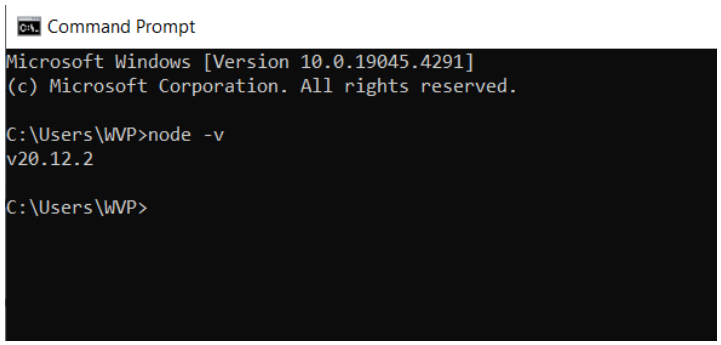

Node.js Setup
Tools for Native Modules
Optionally install the tools necessary to compile native modules.
Some npm modules need to be compiled from C/C++ when installing. If you want to be able to install such modules, some tools (Python and Visual Studio Build Tools) need to b...
Automatically install the necessary tools. Note that this will also install Chocolatey. The script will pop-up in a new window after the installation completes.
Alternatively, follow the instructions at https://github.com/nodejs/node-gyp#on-windows to install the dependencies yourself.
Back
Next
Cancel


Node.js Setup
Completed the Node.js Setup Wizard
Click the Finish button to exit the Setup Wizard.
Node.js has been successfully installed.
Back
Finish
Cancel

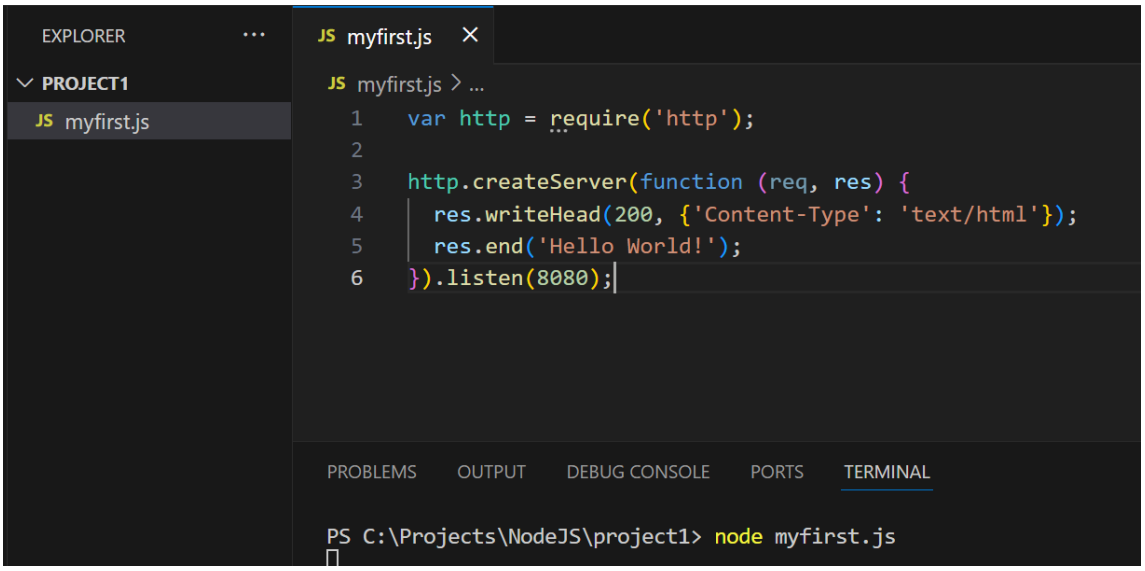
NODEJS BASIC

Version

```
Node -v
```

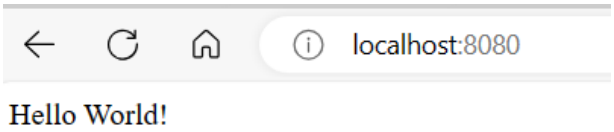


Hello World



Run

```
node myfirst.js
```



GLOBALS

https://youtu.be/OIBIXYLJjsI?list=PL4cUxeGkcC9jsz4LDYc6kv3ymONOKxwBU

Node.js Modules

Create Your Own Modules

The following example creates a module that returns a date and time object:

```
exports.myDateTime = function () {
  return Date();
};
```

Use the **exports** keyword to make properties and methods available outside the module file. Save the code above in a file called "**myfirstmodule.js**"

Include Your Own Module

To include a module, use the **require()** function with the name of the module

Use the module "myfirstmodule" in a Node.js file:

```
var http = require('http');
var dt = require('./myfirstmodule');
http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  res.write("The date and time are currently: " + dt.myDateTime());
  res.end();
}).listen(8080);
```

Notice that we use ./ to locate the module, that means that the module is located in the same folder as the Node.js file.

Another Example:

Creating Module

/Filename.js

```
EXPLORER
PROJECT1
  modules
    people.js
  app.js

app.js   people.js
modules > people.js > ...
1  const people = ['sas','ddsds', 'dsda'];
2
3  console.log(people);
```

Importing Module

```
EXPLORER
PROJECT1
  modules
    people.js
  app.js

app.js   people.js
app.js > ...
1  const man = require('./modules/people');
2
3

PROBLEMS   OUTPUT   DEBUG CONSOLE   PORTS   POLYGLOT NOTEBOOK   TERMINAL

PS C:\Projects\NodeJS\project1> node app
[ 'sas', 'ddsds', 'dsda' ]
PS C:\Projects\NodeJS\project1>
```

Accessing Variable inside Module

Exports

```
EXPLORER
PROJECT1
  modules
    people.js
  app.js

app.js
app.js > ...
1  const man = require('./modules/people');
2
3  console.log(man);

people.js
modules > people.js > ...
1  const people = ['sas','ddsds', 'dsda'];
2
3  console.log(people);
4
5  module.exports = people;

PROBLEMS   OUTPUT   DEBUG CONSOLE   PORTS   POLYGLOT NOTEBOOK   TERMINAL

PS C:\Projects\NodeJS\project1> node app
[ 'sas', 'ddsds', 'dsda' ]
PS C:\Projects\NodeJS\project1> node app
[ 'sas', 'ddsds', 'dsda' ]
[ 'sas', 'ddsds', 'dsda' ]
PS C:\Projects\NodeJS\project1>
```

Exporting Multiple Properties

```
EXPLORER
PROJECT1
  modules
    people.js
  app.js

app.js
const man = require('./modules/people');

console.log(man.people);
console.log(man.ages);

people.js
const people = ['sas','ddsds', 'dsda'];
const ages = [21,25,18];

console.log(people);

module.exports = {
    people : people,
    ages : ages
};
// or
// module.exports = { people, ages };

PROBLEMS  OUTPUT  DEBUG CONSOLE  PORTS  POLYGLOT NOTEBOOK  TERMINAL

PS C:\Projects\NodeJS\project1> node app
[ 'sas', 'ddsds', 'dsda' ]
[ 'sas', 'ddsds', 'dsda' ]
[ 21, 25, 18 ]
PS C:\Projects\NodeJS\project1>
```

Extracting Multiple Properties

```
EXPLORER
PROJECT1
  modules
    people.js
  app.js

app.js
const { people, ages } = require('./modules/people');

console.log(people);
console.log(ages);

people.js
const people = ['sas','ddsds', 'dsda'];
const ages = [21,25,18];

console.log(people);

module.exports = { people, ages };

PROBLEMS  OUTPUT  DEBUG CONSOLE  PORTS  POLYGLOT NOTEBOOK  TERMINAL

PS C:\Projects\NodeJS\project1> node app
[ 'sas', 'ddsds', 'dsda' ]
[ 'sas', 'ddsds', 'dsda' ]
[ 21, 25, 18 ]
PS C:\Projects\NodeJS\project1>
```

Built-in Modules

Node.js has a set of built-in modules which you can use without any further installation.

https://www.w3schools.com/nodejs/ref_modules.asp

File System

The Node.js file system module allows you to work with the file system on your computer.

To include the File System module, use the **require()** method:

```
var fs = require('fs');
```

Common use for the File System module:

- Read files
- Create files
- Update files
- Delete files
- Rename files

https://www.w3schools.com/nodejs/nodejs_filesystem.asp

Reading Files

```
fs.readFile('./docs/blog1.txt', (err, data) => {
    if (err){
        console.log(err);
    }
    console.log(data.toString());
  });
```

Writing Files

```
// writing files (replace existing text inside the file, create file if not exist)
fs.writeFile('./docs/blog1.txt', 'Hello', () => {
    console.log('file was written');
});
```

Directories

Create

```
// create directories
//if not exist
if(!fs.existsSync('./assets')){
    // then create
    fs.mkdir('./assets', (err) => {
        if(err){
            console.log(err);
        }
        console.log('folder created');
    });
}
```

Remove

```
// remove directories
// check if existing
if(fs.existsSync('./assets')){
    // then remove
    fs.rmdir('./assets', (err) => {
        if(err){
```

```
            console.log(err);
        }
        console.log('folder deleted');
    });
}
```

Deleting Files

```
// deleting files
if(fs.existsSync('./docs/deleteme.txt')){
    fs.unlink('./docs/deleteme.txt', (err) => {
        if(err){
            console.log(err);
        }
        console.log('file deleted');
    });
}
```

Streams

Start using the data, before it has finished loading.

https://youtu.be/OIBIXYLJjsI?list=PL4cUxeGkcC9jsz4LDYc6kv3ymONOKxwBU

```
const fs = require('fs');
const readStream = fs.createReadStream('./docs/blog3.txt', { encoding: 'utf8'});
const writeStream = fs.createWriteStream('./docs/blog4.txt');
// using Stream
readStream.on('data', (chunk) => {
    console.log('--- NEW CHUNCK---');
    console.log(chunk);
    writeStream.write('\nNEW CHUNCK\n')
    writeStream.write(chunk)
});
// OR using PIPING
readStream.pipe(writeStream);
```

HTTP Module

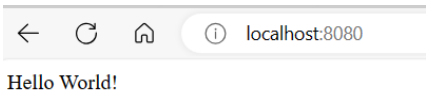
To include a module, use the **require()** function with the name of the module:

```
var http = require('http');
```

Now your application has access to the HTTP module, and can create a server:

```
var http = require('http');

http.createServer(function (req, res) {
  res.end('Hello World!');
}).listen(8080);
```

The function passed into the http.createServer() method, will be executed when someone tries to access the computer on port 8080.



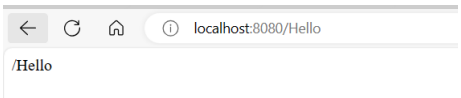
HTTP Header

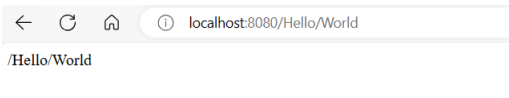
```
var http = require('http');

http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  res.end('Hello World!');
}).listen(8080);
```

The first argument of the **res.writeHead()** method is the status code, 200 means that all is OK, the second argument is an object containing the response headers.

Read the Query String

```
var http = require('http');

http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  res.write(req.url);
  res.end();
}).listen(8080);
```

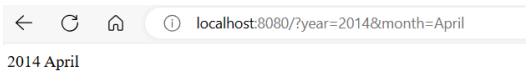




Split Query String

There are built-in modules to easily split the query string into readable parts, such as the URL module.

```
var http = require('http');
var url = require('url');

http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  var q = url.parse(req.url, true).query;
  var txt = q.year + " " + q.month;
  res.end(txt);
}).listen(8080);
```

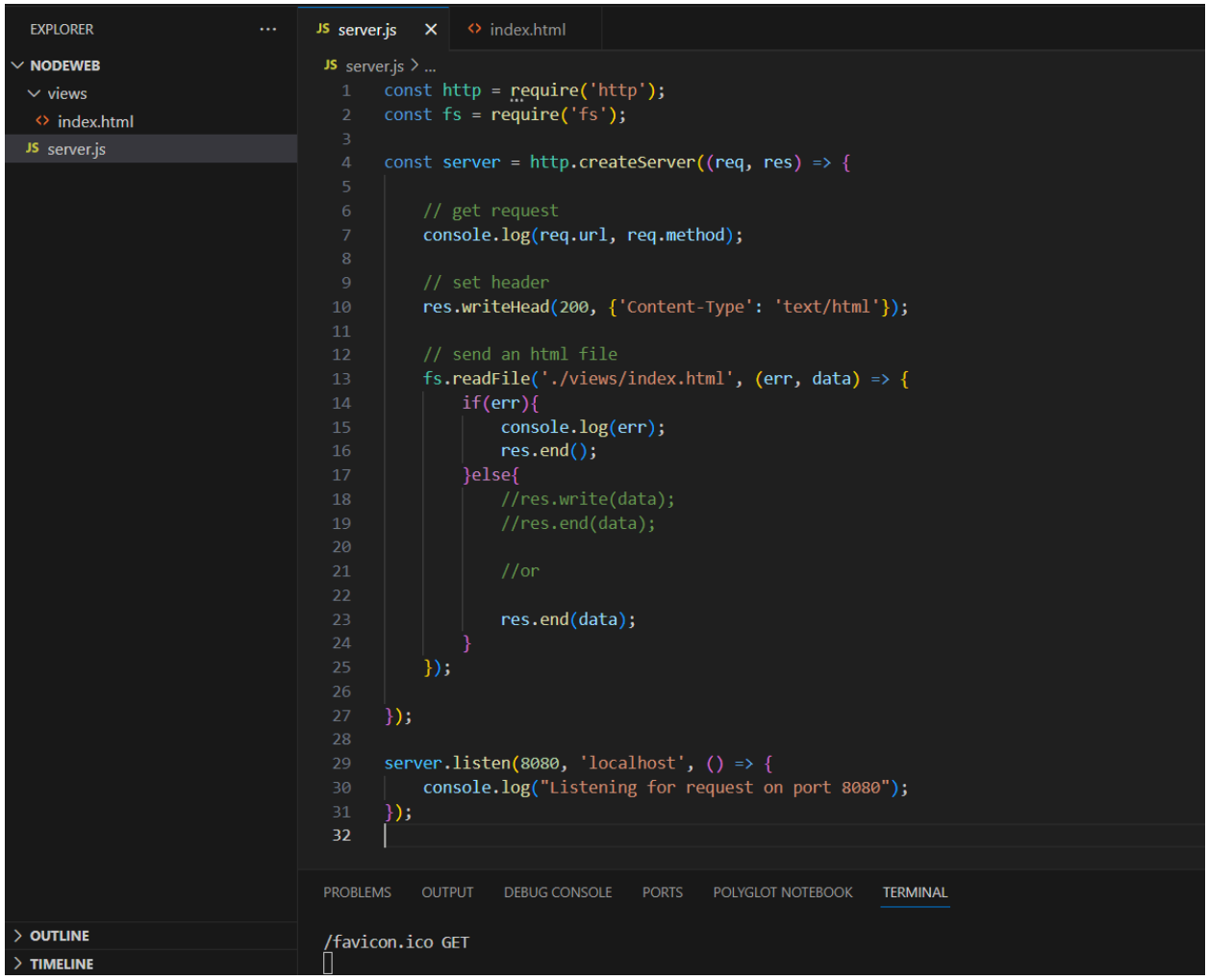


Request & Response

https://youtu.be/DQD00NAUPNk

```
var http = require('http');
const server = http.createServer((req, res) => {
    // get request
    console.log(req.url, req.method);
    // set header
    res.writeHead(200, {'Content-Type': 'text/html'});
    res.write('<h1> Hello </h1>');
    res.write('<h2> World </h2>');
    res.end();
});
server.listen(8080, 'localhost', () => {
    console.log("Listening for request on port 8080");
});
```

HTML




```
const http = require('http');
const fs = require('fs');
const server = http.createServer((req, res) => {
    // get request
    console.log(req.url, req.method);
    // set header
    res.writeHead(200, {'Content-Type': 'text/html'});
    // send an html file
    fs.readFile('./views/index.html', (err, data) => {
        if(err){
            console.log(err);
            res.end();
        }else{
            //res.write(data);
            //res.end(data);
            //or
            res.end(data);
```

```
        }
    });
});
server.listen(8080, 'localhost', () => {
    console.log("Listening for request on port 8080");
});
```

Routing

```
const http = require('http');
const fs = require('fs');
const server = http.createServer((req, res) => {
    // get request
    console.log(req.url, req.method);
    // set header
    res.writeHead(200, {'Content-Type': 'text/html'});

    //ROUTING
    let path = './views/';
    switch(req.url)
    {
        case '/':
            path += 'index.html';
            break;
        case '/about':
            path += 'about.html';
            break;
        default:
            path += '404.html';
            break;
    }

    // send an html file
    fs.readFile(path, (err, data) => {
        if(err){
            console.log(err);
            res.end();
        }else{
            //res.write(data);
            //res.end(data);
            //or
            res.end(data);
        }
    });
});
server.listen(8080, 'localhost', () => {
    console.log("Listening for request on port 8080");
});
```

Status Code

Describe the type of response sent to the browser.

- 200 – OK
- 301 – Resource Moved (Permanent Redirect)
- 404 – Not Found
- 500 – Internal Server Error
- 100 Range – Informational Responses
- 200 Range – Success Codes
- 300 Range – Codes for Redirects
- 400 Range – User or Client Error Codes
- 500 Range – Server Error Codes

```
let path = './views/';
    switch(req.url)
    {
        case '/':
            path += 'index.html';
            res.statusCode = 200;
            break;
        case '/about':
            path += 'about.html';
            res.statusCode = 200;
            break;
        default:
            path += '404.html';
            res.statusCode = 404;
            break;
    }
```

Redirects

```
case '/about-us':
            res.statusCode = 301;
            res.setHeader('Location', '/about');
            res.end();
            break;
```

URL Module

https://www.w3schools.com/nodejs/nodejs_url.asp

NPM

https://youtu.be/bdHE2wHT-gQ

NPM is a package manager for Node.js packages, or modules if you like. www.npmjs.com hosts thousands of free packages to download and use. The NPM program is installed on your computer when you install Node.js.

```
npm init
```

This command create package.json file, a list of dependencies.

Installing Package

A package in Node.js contains all the files you need for a module. Modules are JavaScript libraries you can include in your project.

Downloading a package is very easy. Open the command line interface and tell NPM to download the package you want.

Repository: https://www.npmjs.com/

For Example:

I want to download a package called "nodemon":

```
C:\Users\Your Name>npm install -g nodemon
```

For this example, the package will be installed globally because of using ***-g*** in command. Meaning npm did not install in your project.

For installing packages inside your project. Just install using the command "npm install package_name"

Example:

```
C:\Users\Your Name>npm install upper-case
```

Now you have downloaded and installed your first package! NPM creates a folder named "node_modules", where the package will be placed. All packages you install in the future will be placed in this folder.

My project now has a folder structure like this:

C:\Users\My Name\node_modules\upper-case

Using a Package

Once the package is installed, it is ready to use. Include the "upper-case" package the same way you include any other module:

```
var uc = require('upper-case');
```

Installing Dependencies

```
npm install
```

This command looks at package.json file and install all dependencies listed on that.

Events

Upload Files

Email

MYSQL

Database

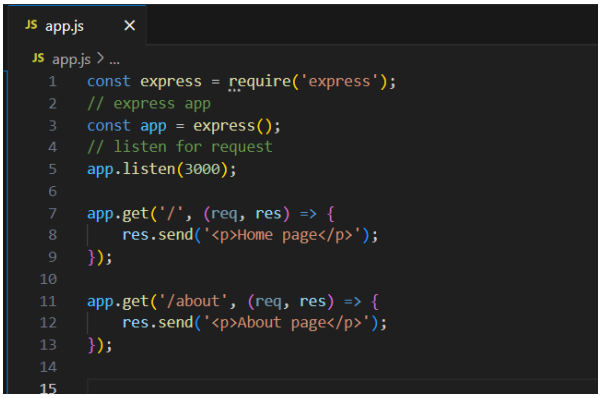
```
const mysql = require('mysql');
const db = mysql.createConnection({
    host: 'localhost',
    user: 'root',
```

```
    password: '',
    database: 'expressdb'
});
db.connect((err) => {
    if (err) throw err;
    db.query("SELECT * FROM customers", function (err, result, fields) {
        if (err) throw err;
        console.log(result);
    });
});
```

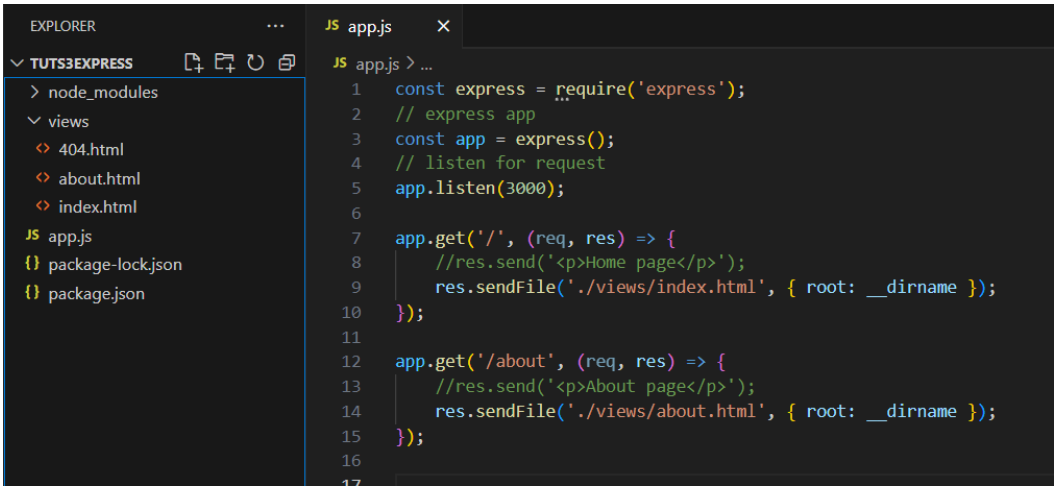
EXPRESSJS

Installation

```
npm install express
```



Basic




```
const express = require('express');
```

```
// express app
const app = express();
// listen for request
app.listen(3000);

app.get('/', (req, res) => {
    //res.send('<p>Home page</p>');
    res.sendFile('./views/index.html', { root: __dirname });
});

app.get('/about', (req, res) => {
    //res.send('<p>About page</p>');
    res.sendFile('./views/about.html', { root: __dirname });
});
```

Redirect and 404

```
// redirect
app.get('/about-us', (req, res) => {
    res.redirect('/about');
});
```

```
// 404
app.use((req, res) => {
    res.sendFile('./views/404.html', { root: __dirname });
});
```

```
// 404
app.use((req, res) => {
    res.status(404).sendFile('./views/404.html', { root: __dirname });
});
```

Template Engine

EJS

Installation

```
npm install ejs
```

EJS automatically see the views inside the **views** folder.

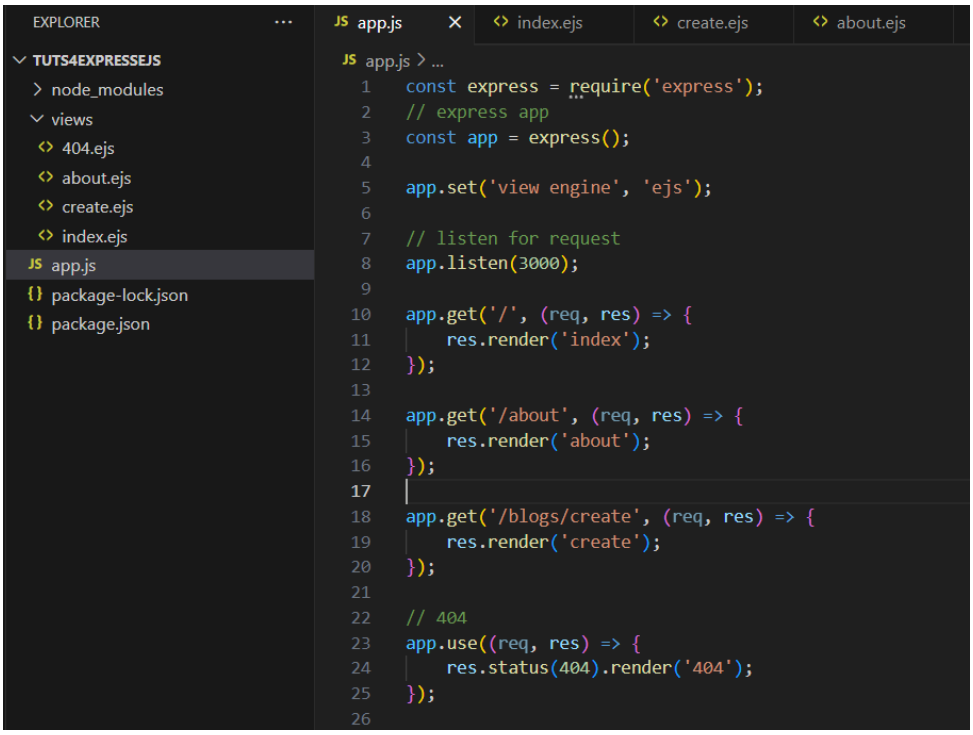
```
const express = require('express');
// express app
const app = express();

app.set('view engine', 'ejs');
```

If the view files are inside **other folder**. Specify the folder name.

```
app.set('views', 'myviews');
```

Render

```
app.get('/', (req, res) => {
    res.render('index');
});
```



Passing Data into Views

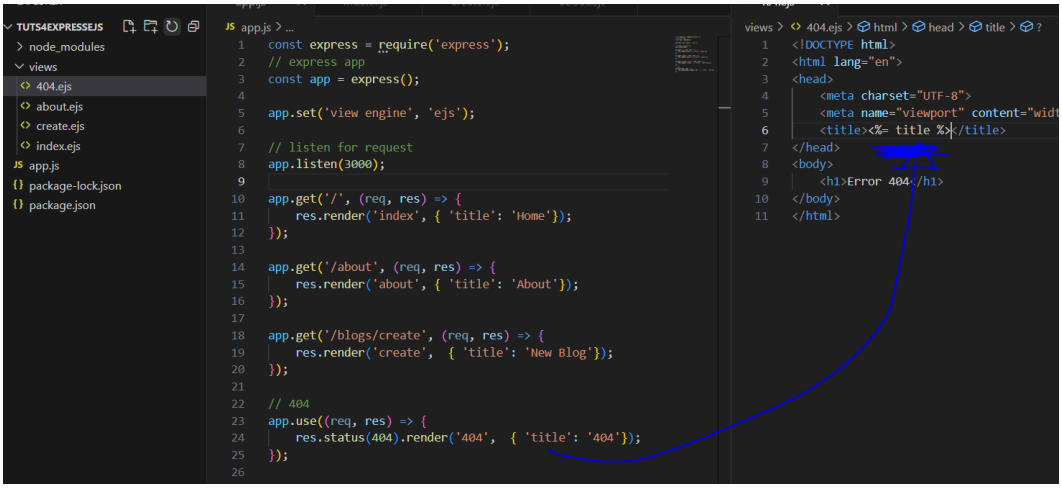
```
res.render('index', { 'title': 'Home'});
```

Render in Views

<title><%= title %></title>

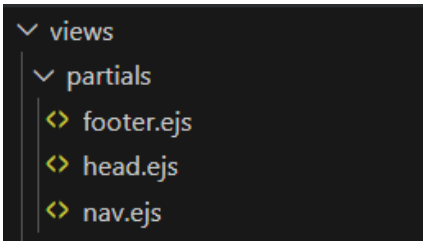
EJS Docummentation

https://ejs.co/#docs



Partials

Create a folder inside '**views**' folder name '**partials'.** Create a file for your partials.



Include

<%- include('./partials/nav.ejs') %>

Middleware

```
app.js > ...
const express = require('express');
// express app
const app = express();

app.set('view engine', 'ejs');

// listen for request
app.listen(3000);

// sample middleware
app.use((req, res) => {
    console.log('host', req.hostname);
    console.log('path', req.path);
    console.log('method', req.method);
});


app.get('/', (req, res) => {
    res.render('index', { 'title': 'Home'});
});

app.get('/about', (req, res) => {
    res.render('about', { 'title': 'About'});
});

app.get('/blogs/create', (req, res) => {
```

The browser hang at the app.use() function.

Next()

```
app.use((req, res, next) => {
    console.log('host', req.hostname);
    console.log('path', req.path);
    console.log('method', req.method);
    next();
});
```

```js
app.js > ...
const express = require('express');
// express app
const app = express();

app.set('view engine', 'ejs');

// listen for request
app.listen(3000);

// sample middleware
app.use((req, res, next) => {
    console.log('host', req.hostname);
    console.log('path', req.path);
    console.log('method', req.method);
    next();
});

// next middleware
app.use((req, res, next) => {
    console.log('another middleware');
    next();
});

app.get('/', (req, res) => {
    res.render('index', { 'title': 'Home'});
});

app.get('/about', (req, res) => {
```

3rd Party Middleware

Example:

```
pip install morgan
```

Usage:

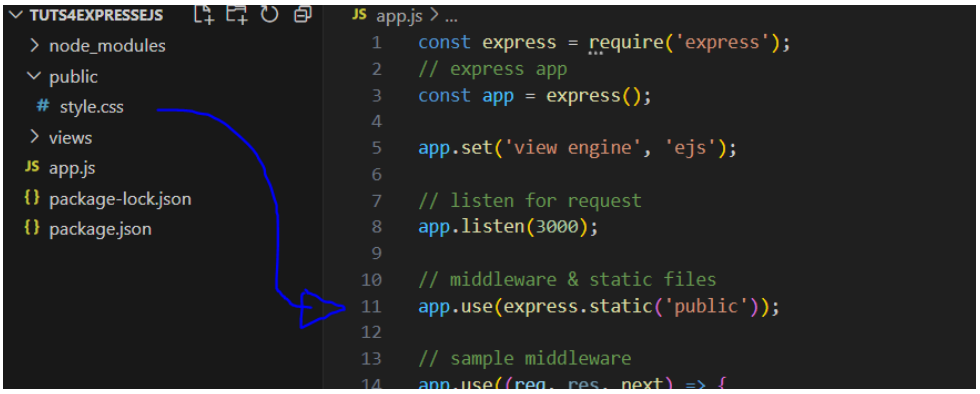
```
const morgan = require('morgan');
app.use(morgan('dev'));
```

Static Files

```
// middleware & static files
app.use(express.static('public'));
```

or

```
const path = require('path');
// Set path for static files (e.g., CSS, images)
app.use(express.static(path.join(__dirname, 'public')));
```




```
<link rel="stylesheet" href="/style.css">
```

Get, Post, Delete Request

- Get – get resources.
- Post – create new data.
- Delete – delete data.
- Put – update data.

Get Request

```
app.get('/about', (req, res) => {
    res.render('about', { 'title': 'About'});
});
```

Accepting Form Data

```
app.use(express.urlencoded({ extended: true }));
```

OR

```
const bodyParser = require('body-parser');
// Parse application/x-www-form-urlencoded
app.use(bodyParser.urlencoded({ extended: false }));
// Parse application/json
app.use(bodyParser.json());
```

Post Request

```
app.post('/blogs/create', (req, res) => {
    console.log(req.body);
});
```

Route Parameters

The variable parts of the route that may change value.

localhost:3000/blogs/:id

localhost:3000/blogs/123

localhost:3000/blogs/25

Example:

```
app.get('/blogs/:id', (req, res) => {
    const id = req.params.id;
    console.log(id);
});
```

Delete

View Code:

```
<a class="delete" data-doc="<%= blog._id %>">delete</a>
<script>
    const trashcan = document.querySelector('a.delete');
    trashcan.addEventListener('click', (e) => {
        const endpoint = `/blogs/${trashcan.dataset.doc}`;
        fetch(endpoint, {
            method: 'DELETE'
        })
        .then(() => {})
        .catch(err => console.log(err));
    });
</script>
```

```
<a class="delete" data-doc="<%= blog._id %>">delete</a>
<script>
    const trashcan = document.querySelector('a.delete');
    trashcan.addEventListener('click', (e) => {
        const endpoint = `/blogs/${trashcan.dataset.doc}`;
        fetch(endpoint, {
            method: 'DELETE'
        })
        .then(() => {})
        .catch(err => console.log(err));
    });
</script>
```

App Code

```
app.delete('/blogs/:id', (req, res) => {
    const id = req.params.id;
    console.log(id, ' is deleted');
});
```

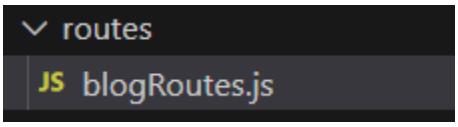
Continue at https://youtu.be/VVGgacjzc2Y?t=1848

Express Router & MVC

Express Router

Split the route in different files.

Create a folder "**routes**" and create a **routeBlog.js**




```
const express = require('express');
// express app
const router = express.Router();
// Example of using route
router.get('/blogs/create', (req, res) => {
    res.render('create',  { 'title': 'New Blog'});
});
//export router
module.exports = router;
```

On your app.js file, import your router:

```
const blogRoute = require('./routes/blogRoutes');
```

Using your routes

```
app.use(blogRoute);
```

OR use this

```
app.use('/blog', blogRoute);
```

http://localhost/blog/blogRoutes

EXPLORER
TUTS4EXPRESSEJS...
- node_modules
- public
- routes
  - blogRoutes.js
- views
- app.js
- package-lock.json
- package.json

app.js > app.use() callback

```js
const express = require('express');
const blogRoute = require('./routes/blogRoutes');
// express app
const app = express();

app.set('view engine', 'ejs');

// listen for request
app.listen(3000);

// middleware & static files
app.use(express.static('public'));
app.use(express.urlencoded({ extended: true }));

// sample middleware
app.use((req, res, next) => {
    console.log('host', req.hostname);
    console.log('path', req.path);
    console.log('method', req.method);
    next();
});

// next middleware
app.use((req, res, next) => {
    console.log('another middleware');
    next();
});

app.get('/', (req, res) => {
    res.render('index', { 'title': 'Home'});
});

app.get('/about', (req, res) => {
    res.render('about', { 'title': 'About'});
});

app.use(blogRoute);

// 404
app.use((req, res) => {
    res.status(404).render('404',  { 'title': '404'});
});
```

blogRoutes.js > ...

```js
const express = require('express');

// express app
const router = express.Router();

router.get('/blogs/create', (req, res) => {
    res.render('create',  { 'title': 'New Blog'});
});

router.post('/blogs/create', (req, res) => {
    console.log(req.body);
});

router.get('/blogs/:id', (req, res) => {
    const id = req.params.id;
    console.log(id);
});

router.delete('/blogs/:id', (req, res) => {
    const id = req.params.id;
    console.log(id, ' is deleted');
});

module.exports = router;
```

EXPLORER
TUTS4EXPRESSEJS...
- node_modules
- public
- routes
  - blogRoutes.js
- views
- app.js
- package-lock.json
- package.json

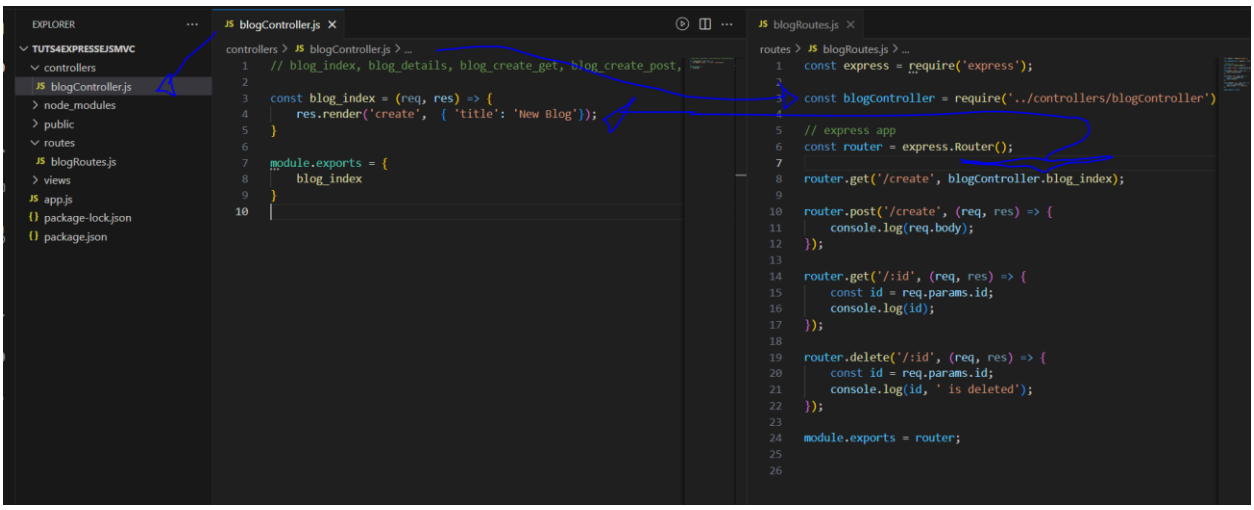
app.js > app.get('/') callback

```js
const express = require('express');
const blogRoute = require('./routes/blogRoutes');
// express app
const app = express();

app.set('view engine', 'ejs');

// listen for request
app.listen(3000);

// middleware & static files
app.use(express.static('public'));
app.use(express.urlencoded({ extended: true }));

// sample middleware
app.use((req, res, next) => {
    console.log('host', req.hostname);
    console.log('path', req.path);
    console.log('method', req.method);
    next();
});

// next middleware
app.use((req, res, next) => {
    console.log('another middleware');
    next();
});

app.get('/', (req, res) => {
    res.render('index', { 'title': 'Home'});
});

app.get('/about', (req, res) => {
    res.render('about', { 'title': 'About'});
});

app.use('/blogs', blogRoute);

// 404
app.use((req, res) => {
    res.status(404).render('404',  { 'title': '404'});
});
```

blogRoutes.js > ...

```js
const express = require('express');

// express app
const router = express.Router();

router.get('/create', (req, res) => {
    res.render('create',  { 'title': 'New Blog'});
});

router.post('/create', (req, res) => {
    console.log(req.body);
});

router.get('/:id', (req, res) => {
    const id = req.params.id;
    console.log(id);
});

router.delete('/:id', (req, res) => {
    const id = req.params.id;
    console.log(id, ' is deleted');
});

module.exports = router;
```

MVC



Create a folder "controllers" and add **blogController.js** file

blogController.js

```
const blog_index = (req, res) => {
    res.render('create',  { 'title': 'New Blog'});
}

module.exports = {
    blog_index
}
```

Import blogController into the route

```
const blogController = require('../controllers/blogController');
```

From this:

```
router.get('/create', (req, res) => {
    res.render('create',  { 'title': 'New Blog'});
});
```

To this:

```
router.get('/create', blogController.blog_index);
```

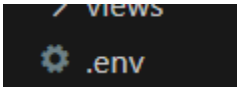
```js
const homeRoutes = require('./routes/homeRoutes');

app.use('/home', homeRoutes);
app.use('/', homeRoutes);
app.use('/scan', scanRoutes);
app.use('/info', infoRoutes);
app.use('/trivia', triviaRoutes);

// 404
app.use((req, res) => {
    res.status(404).render('404', { 'title': '404'});
});
```

```js
const express = require('express');

const homeController = require('../controllers/homeController');

// express app
const router = express.Router();

router.get('/', homeController.home_index);

module.exports = router;
```

```js
//

const home_index = (req, res) => {
    res.render('index', { 'title': 'Home', 'page_name': 'home' });
}

module.exports = {
    home_index
}
```

ENV

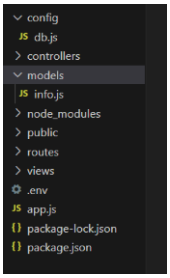
A **.env** file is a hidden text file that is used to pass environment variables to your application. Environment variables are key value pairs of data that you want to keep private or hidden, such as passwords, API keys, or database credentials.




```env
DB_HOST=localhost
DB_USER=root
DB_PASSWORD=
DB_NAME=dbname
```

Using Model

Create folder and file `**config/db.js**` and `**models/model_name.js**`.



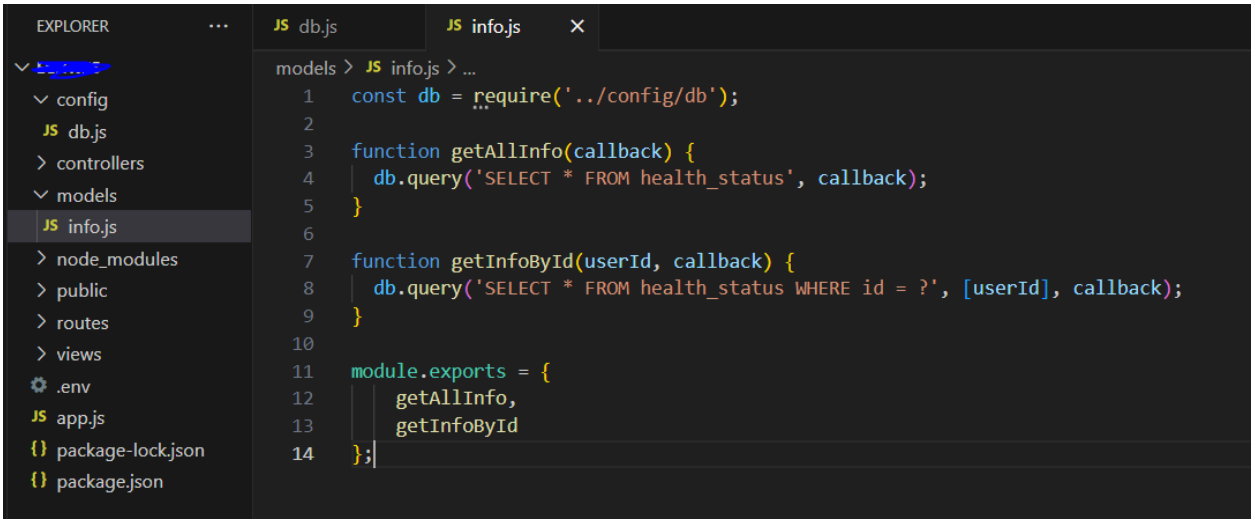
Database Connection:

```
require('dotenv').config();
const mysql = require('mysql2');

const connection = mysql.createConnection({
```

```
    host: process.env.DB_HOST,
    user: process.env.DB_USER,
    password: process.env.DB_PASSWORD,
    database: process.env.DB_NAME
});

connection.connect((err) => {
    if (err) throw err;
    console.log('Connected to MySQL database');
});

module.exports = connection;
```

Model:

```
const db = require('../config/db');

function getAllInfo(callback) {
  db.query('SELECT * FROM health_status', callback);
}

function getInfoById(userId, callback) {
  db.query('SELECT * FROM health_status WHERE id = ?', [userId], callback);
}

module.exports = {
    getAllInfo,
    getInfoById
};
```



Controller:

```
const infoModel = require('../models/info');

const info = (req, res) => {

   infoModel.getAllInfo((err, data) => {
      if (err) {
        console.error('Error retrieving data:', err);
        res.status(500).send('Error retrieving data');
        return;
      }
      //res.json(data);
      res.render('info/index',  { 'info': data,'title': 'Info', 'page_name': 'info' });
   });

}

module.exports = {
   info
}
```

Prevent XSS (Cross-Site Scripting) attacks

html-entities

Install html-entities package:

```
npm install html-entities
```

Update Controller to Render View with Sanitized Data:

```
const { encode } = require('html-entities');
// Sanitize user data
  const sanitizedUsers = users.map(user => {
    return {
      id: user.id,
      name:encode(user.name), // HTML-encode user name
      // Add more fields to sanitize as needed
    };
  });
```

Example:

```
controllers > JS infoController.js > ...
//
const infoModel = require('../models/info');

const { encode } = require('html-entities');

const info = (req, res) => {

    infoModel.getAllInfo((err, data) => {
        if (err) {
          console.error('Error retrieving data:', err);
          res.status(500).send('Error retrieving data');
          return;
        }

        // Sanitize data
        const sanitizedData = data.map(data => {
            return {
            id: data.id,
            name: encode(data.name), // HTML-encode user name
            // Add more fields to sanitize as needed
            description: encode(data.description),
            };
        });

        //res.json(data);
        res.render('info/index',  { 'info': sanitizedData,'title': 'Info', 'page_name': 'info' });
    });

}

module.exports = {
    info
}
```

Prevent SQL Injections

Use Parameterized Queries

Instead of directly embedding user input into SQL queries, use parameterized queries or prepared statements. Parameterized queries separate SQL code from user input, preventing SQL injection attacks. Libraries like mysql2 support parameterized queries.

```
function createUser(username, email) {
  return new Promise((resolve, reject) => {
    const sql = 'INSERT INTO users (username, email) VALUES (?, ?)';
    db.query(sql, [username, email], (err, results) => {
      if (err) {
        reject(err);
        return;
      }
      resolve(results.insertId); // Return the ID of the inserted user
    });
  });
}
```

Validate User Input

Validate user input to ensure it meets expected criteria before saving it to the database. Use validation libraries like validator to sanitize and validate input fields such as email addresses, usernames, and passwords.

```
const validator = require('validator');

function createUser(username, email) {
  if (!validator.isEmail(email)) {
    throw new Error('Invalid email address');
  }
  // Validate other fields as needed
  // ...
}
```

Sanitize User Input

Sanitize user input to remove potentially harmful characters or scripts before saving it to the database. Libraries like xss can help sanitize user input to prevent XSS attacks.

```
const xss = require('xss');

function createUser(username, email) {
  const sanitizedUsername = xss(username);
  const sanitizedEmail = xss(email);
  // Save sanitized data to the database
  // ...
}
```

Escape User Input

If you're manually constructing SQL queries (not recommended), escape user input using functions provided by the database library to prevent SQL injection attacks.

```
const db = require('../config/db');

function createUser(username, email) {
  const escapedUsername = db.escape(username);
  const escapedEmail = db.escape(email);
  // Save escaped data to the database
  // ...
}
```

Session

express-session

```
const express = require('express');
const session = require('express-session');

const app = express();

// Configure session middleware
app.use(session({
  secret: 'your_secret_key',
  resave: false,
  saveUninitialized: false
}));

// Routes
app.get('/', (req, res) => {
  // Access session data
  if (req.session.username) {
    res.send(`Welcome back, ${req.session.username}!`);
  } else {
    res.send('Welcome, please log in.');
  }
});

app.get('/login', (req, res) => {
  // Set session data
  req.session.username = 'user123';
  res.redirect('/');
});

app.get('/logout', (req, res) => {
  // Destroy session data
  req.session.destroy();
  res.redirect('/');
});

app.listen(3000, () => {
  console.log('Server is running on port 3000');
});
```

File System Store

Install Dependencies

```
npm install express-session session-file-store
```

Require Modules

```
const express = require('express');
const session = require('express-session');
const FileStore = require('session-file-store')(session);
```

Configure Session Middleware

```
const app = express();

// Configure session middleware
app.use(session({
  store: new FileStore({
    path: './sessions', // Specify the directory where session data will be stored
    ttl: 86400, // Session expiration time in seconds (optional)
    retries: 0, // Number of retries on failed write (optional)
    logFn: function () {} // Logging function (optional)
  }),
  secret: 'your_secret_key', // Secret key used for session encryption
  resave: false, // Don't save session if unmodified
  saveUninitialized: false // Don't create session until something is stored
}));
```

Use Sessions

```
app.get('/', (req, res) => {
  if (req.session.views) {
    req.session.views++;
    res.send(`You have visited this page ${req.session.views} times.`);
  } else {
    req.session.views = 1;
    res.send('Welcome to this page for the first time!');
  }
});
```

Best Security Practices

https://expressjs.com/en/advanced/best-practice-security.html

ORM

Sequelize

```
npm install sequelize mysql2
```

https://sequelize.org/docs/v6/

Initialize Sequelize:

```
// sequelize.js
const { Sequelize } = require('sequelize');

const sequelize = new Sequelize('database', 'username', 'password', {
  host: 'localhost',
  dialect: 'mysql', // or 'postgres', 'sqlite', 'mssql', etc.
});

module.exports = sequelize;
```

Define Models:

```
// models/User.js
const { DataTypes } = require('sequelize');
const sequelize = require('../sequelize');

const User = sequelize.define('User', {
  // Define model attributes
  firstName: {
    type: DataTypes.STRING,
    allowNull: false
  },
  lastName: {
    type: DataTypes.STRING,
    allowNull: false
  },
  email: {
    type: DataTypes.STRING,
    allowNull: false,
    unique: true
  }
});

module.exports = User;
```

Sync Models with Database

```
// app.js
const sequelize = require('./sequelize');
```

```
const User = require('./models/User');

async function initializeDatabase() {
  try {
    await sequelize.authenticate();
    console.log('Connection to the database has been established successfully.');
    await sequelize.sync(); // Sync all models with the database
    console.log('All models were synchronized successfully.');
  } catch (error) {
    console.error('Unable to connect to the database:', error);
  }
}

initializeDatabase();

// Other app.js code...
```

Use Models in Routes/Controllers:

```
// routes/users.js
const express = require('express');
const router = express.Router();
const User = require('../models/User');

router.get('/', async (req, res) => {
  try {
    const users = await User.findAll();
    res.json(users);
  } catch (error) {
    console.error('Error retrieving users:', error);
    res.status(500).send('Error retrieving users');
  }
});

module.exports = router;
```

Updating dependencies

Updating dependencies in a Node.js project involves using the npm update command. This command updates all the packages listed in the package.json file to their latest versions according to the specified version range.

Here's how you can update all dependencies:

1. Open your terminal or command prompt.
2. Navigate to your project directory: Use the cd command to navigate to the directory containing your project.

   ```
   cd /path/to/your/project
   ```

3. Run the npm update command: Use the following command to update all dependencies to their latest versions:

   ```
   npm update
   ```

   If you want to update only specific packages, you can specify their names after the npm update command.

   ```
   npm update package1 package2 ...
   ```

4. Review the changes: After running the command, npm will display the packages that were updated and their new versions. Review these changes to ensure they are compatible with your project requirements.
5. Test your application: After updating dependencies, it's a good practice to test your application thoroughly to ensure that the updates did not introduce any issues or breaking changes.
6. Commit the changes: If everything looks good, you can commit the changes to your version control system (e.g., Git).

   ```
   git add package.json package-lock.json

   git commit -m "Update dependencies"
   ```

7. Push the changes: If you're working in a collaborative environment, push the committed changes to your remote repository.

   ```
   git push origin <branch-name>
   ```

PostgreSQL

```
npm install --save pg pg-hstore
```

Connecting to a database

```
const { Sequelize } = require('sequelize');

// Option 1: Passing a connection URI
const sequelize = new Sequelize('sqlite::memory:') // Example for sqlite
const sequelize = new Sequelize('postgres://user:pass@example.com:5432/dbname') // Example for postgres

// Option 2: Passing parameters separately (sqlite)
const sequelize = new Sequelize({
  dialect: 'sqlite',
  storage: 'path/to/database.sqlite'
});

// Option 3: Passing parameters separately (other dialects)
const sequelize = new Sequelize('database', 'username', 'password', {
  host: 'localhost',
  dialect: /* one of 'mysql' | 'postgres' | 'sqlite' | 'mariadb' | 'mssql' | 'db2' | 'snowflake' | 'oracle' */
});
```