**What is Express?**

**Express.js** is a **web framework for Node.js**. It helps you build **web servers** and **APIs** easily and efficiently.

**What is Node.js?**

* **Node.js** is a **JavaScript runtime**.
* It allows you to run JavaScript **outside the browser**, like on your **local machine/server**.
* It uses the **V8 engine** (same as Chrome) to execute JavaScript.

Why Use **Express.js** with Node.js?

**Removes Boilerplate Code**

* Native Node requires manual handling of URLs, headers, etc.
* Express handles all of that for you in a cleaner way.

|  |  |
| --- | --- |
| **Without Express:**  const http = require('http');  http.createServer((req, res) => {  if (req.url === '/hello' && req.method === 'GET') {  res.write('Hello World');  res.end();  }  }).listen(3000); | **With Express:**  const express = require('express');  const app = express();  app.get('/hello', (req, res) => {  res.send('Hello World');  });  app.listen(3000); |

**Express is Middleware-Based**

* Middleware is a function that runs **between** the request and response.
* You can use middleware for:
  + Authentication
  + Logging
  + Parsing JSON
  + Error handling

**Routing Made Easy**

Express makes it simple to handle different HTTP methods and paths:

app.get('/', (req, res) => res.send('GET request'));

Setting Up the project:

1. mkdir my-express-app
2. cd my-express-app
3. npm init -y
4. npm install nodemon
5. npm install express
6. start nodemon app.js

**install nodemon**

**Nodemon** is a **utility tool** for Node.js that **automatically restarts your server** when you make changes to your code.

**npm install nodemon**

Import express module:

const express = require('express');

**What does it return?**

It **returns a function** — specifically, the **main Express function**.

const express = require('express');

const app = express(); // calling the function

**So what is app?**

After calling express(), it returns an **Express application object** (called app), which you use to:

✅ Define routes  
✅ Use middleware  
✅ Start the server

What is app.get in Express?

Syntax: app.get(path, callback)

Example: app.get('/', (req, res) => {

res.send('Hello World!'); });

1. app → the Express application
2. .get → handle **GET** requests
3. '/' → the **route path** (homepage)
4. (req, res) → the **callback function** with:
5. req = request object
6. res = response object
7. res.send() → sends response to the client

App.listen

It starts the server and tells Express to **listen for incoming connections** on the specified **port**

app.listen(3000, () => {

console.log('Server is running on http://localhost:3000');

});

MiddleWare In Express:

const express = require('express');

const app = express();

// 1. Home Route

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

res.send('Welcome to Home Page');

});

// 2. Middleware function

function loginMiddleware(req, res, next) {

console.log('I am the login middleware');

// If next() is NOT called, request hangs

next(); // pass control to next route or middleware

}

// 3. Apply middleware to all routes below

app.use(loginMiddleware);

// 4. Protected route

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

res.send('Welcome to your profile!');

});

// 5. Start server

app.listen(3000, () => {

console.log('Server is running on http://localhost:3000');

});

**Why is next() Important in Middleware?**

Middleware functions in Express must do **one of three things**:

1. Call next() → pass to the next route or middleware ✅
2. Send a response → res.send(), res.json(), etc.
3. End the response → res.end(), res.redirect()

**If you don't call next() and don't send a response:**

function loginMiddleware(req, res, next) {

console.log('I am the login middleware');

// next(); <-- if you forget this

}

The request hangs forever → your page just **keeps loading** because Express doesn't know what to do next.

//basic example of middleware

function loginMiddleware(req, res, next) {

console.log('Checking login...');

// Simulate a login check (fake user is logged in)

const userIsLoggedIn = true;

if (userIsLoggedIn) {

next(); // Continue to /profile route

} else {

res.status(401).send('Unauthorized: Please log in');

}

}

app.listen(PORT) Statement

const express = require('express');

const app = express();

const PORT = 3000;

app.listen(PORT, () => {

console.log(`Server running on port ${PORT}`);

});(backticks `, not quotes), and ${PORT} is how you inject a variable into a string using **JavaScript's template literals** (introduced in ES6).

PACKAGE JSON :

"scripts": {

"start": "node server.js",

"dev": "nodemon server.js",

}

**1."start": "node server.js"**

* This tells Node.js to **run your app** by executing the file server.js.

Run this by the command:

npm start

Note: "start" is special. You can run it **without** typing run.

**2. "dev": "nodemon server.js"**

* Runs your app using nodemon, which **watches for file changes** and **automatically restarts the server**.
* Run with: npm run dev

**You can also specify start:nodemon server.js**

**App.get vs router.get**

You write everything directly in your main file:

|  |  |
| --- | --- |
| // server.js  const express = require('express');  const app = express();  app.get('/', (req, res) => {  res.send('🏠 Home Page');  });  app.get('/about', (req, res) => {  res.send('ℹ️ About Page');  });  app.listen(3000, () => {  console.log('Server running on port 3000');  });  Everything is handled directly by app (your main Express app). | // routes/userRoutes.js  const express = require('express');  const router = express.Router();  router.get('/profile', (req, res) => {  res.send('👤 User Profile');  });  router.get('/settings', (req, res) => {  res.send('⚙️ User Settings');  });  module.exports = router;  // server.js  const express = require('express');  const app = express();  const userRoutes = require('./routes/userRoutes'); // import router  app.use('/user', userRoutes); // mount router at /user  app.listen(3000, () => {  console.log('Server running on port 3000');  }); |

| **app.get()** | **router.get()** |
| --- | --- |
| Used in server.js directly | Used in a separate file (router) |
| Good for small apps | Good for **large apps** (organized) |
| Everything is in one place | You can **split routes** into files |

Hey if u use app.get you need to define all routes in server.js it is not suitable for large application so u used router.get in separate file named useRoutes.js where you reach that by

const userRoutes = require('./routes/userRoutes'); // import router

app.use('/user', userRoutes); // mount router at /user

when your ulr starts with user it forward to that file

localhost:3000/user/about

so here the url starts with user so it forward to userRoutes.js where you defined the route for router.get(/about,()=>{});

Chained Routes:

router.route("/").get(users).post(create);

* When someone sends a **GET** request to /, Express will run the users function.
* When someone sends a **POST** request to /, Express will run the create function.

It is equivalent to writing :

router.get("/", users);

router.post("/", create);

|  |  |
| --- | --- |
| const express = require('express');  const router = express.Router();  // Controller functions  const users = (req, res) => {  res.send('👥 Here are all users');  };  const create = (req, res) => {  res.send('✅ User created');  };  // Route chaining  router.route("/").get(users).post(create);  module.exports = router; | const express = require('express');  const app = express();  const userRoutes = require('./userRoutes');  app.use('/users', userRoutes); |

What is Error-Handling Middleware?

(err, req, res, next) => { ... }

**Why use it?**

To **centralize** error handling and avoid writing try-catch everywhere.  
It catches errors passed using next(err).

Whenever error arise in one route it send the error to some middleware using next(errcontent) The middleware which has this four parameter will receive this and process the errror

app.post('/users', (req, res, next) => {

const { name } = req.body;

if (!name) {

return next(new Error('Name is required'));

}

res.send('User created');

});

**if name is missing, the error will be passed to:**

app.use((err, req, res, next) => {

res.status(400).json({ error: err.message });

});

**How It Works:**

1. If there's an error inside any route or middleware...
2. You pass it using next(err)
3. Express will **skip all normal middleware** and go directly to the error-handling middleware.

**Types of Middleware**

ExpressJS offers different types of middleware and you should choose the middleware based on functionality required.

* Application-level middleware: Bound to the entire application using [app.use()](https://www.geeksforgeeks.org/express-js-app-use-function/) or [app.METHOD()](https://www.geeksforgeeks.org/express-js-app-method-function/) and executes for all routes.
* Router-level middleware: Associated with specific routes using[router.use()](https://www.geeksforgeeks.org/express-js-router-use-function/) or [router.METHOD()](https://www.geeksforgeeks.org/express-js-router-method-function/) and executes for routes defined within that router.
* Error-handling middleware: Handles errors during the request-response cycle. Defined with four parameters (err, req, res, next).
* Built-in middleware: Provided by Express (e.g., express.static, ExpressJSon, etc.).
* Third-party middleware: Developed by external packages (e.g., body-parser, morgan, etc.).

Built In Middleware:

Express provides built-in middleware to handle common tasks. For instance, express.json() and express.urlencoded() parse JSON and URL-encoded data

app.use(express.json());

This is a **built-in middleware** in Express that tells your app:

"Hey, if the client sends me JSON data, I want to **parse it automatically** and make it available in req.body

**Why do we need it?**

When a client (like Postman, frontend, or mobile app) sends data in **JSON format** via POST or PUT, Express doesn’t understand it **by default**.

Without express.json(), this will happen:

app.post('/data', (req, res) => {

console.log(req.body); // ❌ undefined

});

But with this:

app.use(express.json());

app.post('/data', (req, res) => {

console.log(req.body); // ✅ prints the parsed JSON object

});

Commonly used:

app.use(express.json()); // For JSON

app.use(express.urlencoded({ extended: true })); // For form data that is html & extended: true lets it handle nested objects

app.use(express.static('public')); // Serve static files

app.all(\*):/ this is used at the end if no routes match the requested url it will execute and send 404 not found instead of throwing error

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

res.send('Welcome home!');

});

app.all('\*', (req, res) => {

res.status(404).send('404 - Not Found');

});

**What is Morgan?**

**Morgan** is an **HTTP request logger middleware** for Node.js/Express.

Think of it as a tool that automatically logs **incoming requests** in the console — great for debugging and development.

**Why use Morgan?**

Because it logs info like:

* HTTP method (GET, POST, etc.)
* URL requested
* Status code
* Response time
* Timestamp

app.use(morgan('dev')); // log requests to the console

| **Format** | **Description** | **Output Example** |
| --- | --- | --- |
| 'dev' | Colored concise output | GET / 200 8.500 ms - 5 |
| 'tiny' | Minimal output | GET / 200 - - |
| 'combined' | Apache-style logs (good for production) | Logs IP, method, URL, status, user-agent, etc. |
| 'common' | Standard Apache common log output | Similar to combined without referrer & UA |

Mongo DB:

NPM INSTALL MONGODB

* **MongoDB** is like a **blank notebook** – you can write anything in it.
* **Mongoose** is like a **form/template** – it makes sure you write data in a proper format.

Mongodb:

**No schema** (schema-less)

Mongoose:

Uses **schemas** for structure

**What is body-parser?**

* It’s a **middleware** for Express.
* It **parses the incoming request body** and makes it available under req.body

Example:

const express = require('express');

const bodyParser = require('body-parser');

const app = express();

app.use(bodyParser.json()); // 👈 This parses JSON bodies

app.post('/user', (req, res) => {

console.log(req.body); // ✅ Works!

res.send('Data received');

});

// Middleware to handle URL-encoded data

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

<form action="/submit" method="POST">

<input name="name" value="Alice" />

<input name="email" value="alice@example.com" />

<button type="submit">Send</button>

</form>name=Alice&email=alice%40example.com this magic urlencoder helps to understand this and make available

Connections:

const mongoose = require('mongoose');

mongoose.connect('mongodb+srv://myUser:myPass@cluster0.mongodb.net/myAppDB?retryWrites=true&w=majority', {

useNewUrlParser: true,

useUnifiedTopology: true,

})

.then(() => console.log('✅ Connected to MongoDB Atlas'))

.catch(err => console.error('❌ Connection error:', err));

const mongoose = require('mongoose');

// Step 1: Create Schema

const userSchema = new mongoose.Schema({

name: {

type: String,

required: true,

},

email: {

type: String,

required: true,

unique: true,

},

age: Number,

});

// Step 2: Create Model

const User = mongoose.model('User', userSchema);

// Step 3: Export the Model

module.exports = User;