PROJECT TRAINING WORKSHOP

Node JS

Node JS

- **What is Node JS?**
  - server-side runtime environment for executing JavaScript code (Runtime + JavaScript)

- **What are the roles of Node JS?**

JavaScript Runtime	Server-Side Programming	Event Driven and Non-Blocking I/O	Scalability
NPM (Node Package Manager)	Single Language (Client and Server)	Rich Community and Ecosystem	Streaming and Data Processing
Cross-Platform	Open Source		

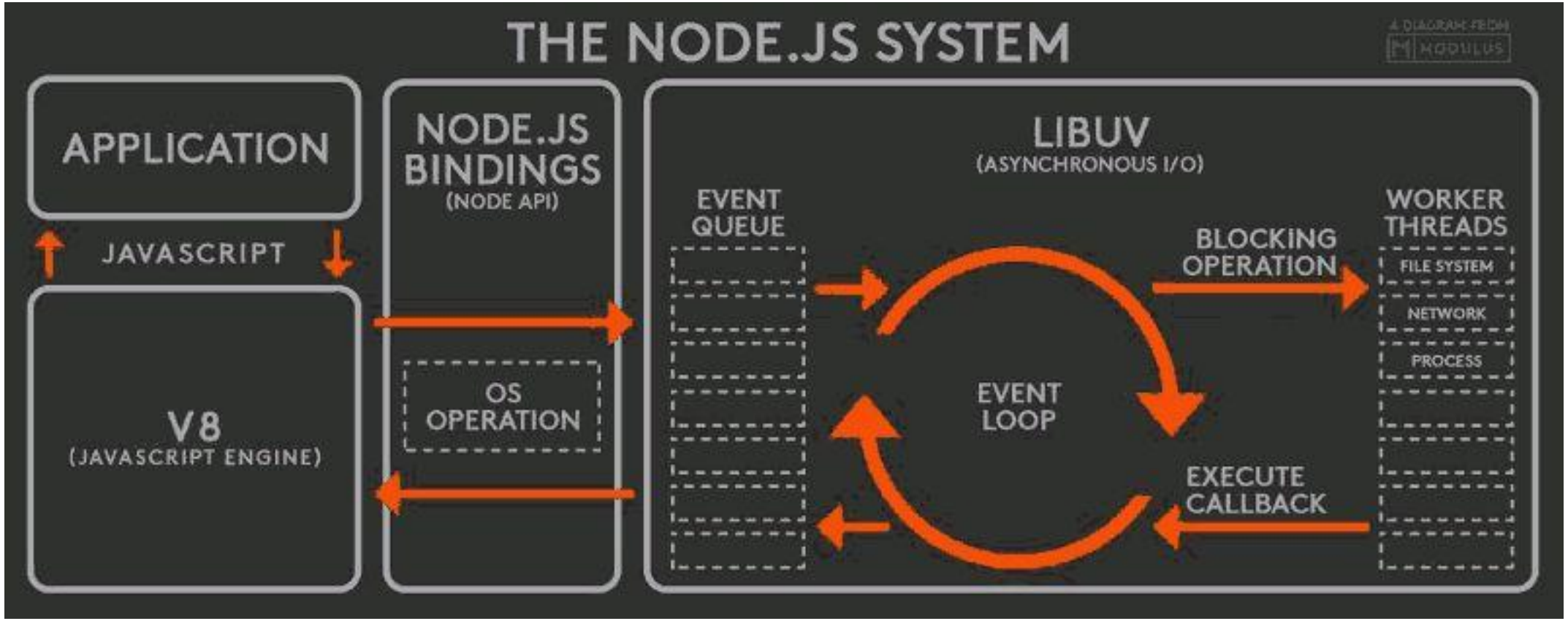
- **What are the roles of Node JS?**
  - **Single Page Applications**
  - **Data Streaming Applications**
  - **Real Time Application (Data Intensive)**
  - **JSON APIs based Applications**
  - **I/O bound Applications**

Node JS

- **What are the key concepts of Node JS?**

V8 JavaScript Engine (google)	Event Loop	Modules ('require')	Node Package Manager (open-source collection)
Asynchronous Programming	Event Emitters (custom event-driven modules)	Streams (files, http requests, data processing)	Built-in HTTP Module (RESTful APIs)
Buffers (Binary Data)	File System Modules (fs)	Promises and Async/Await	Child Processes (parallel code execution)
Global Objects (process, console, require)	Error Handling (try/catch)	Single Threaded	

Node JS ARCHITECTURE



Node JS – Building Blocks of Application

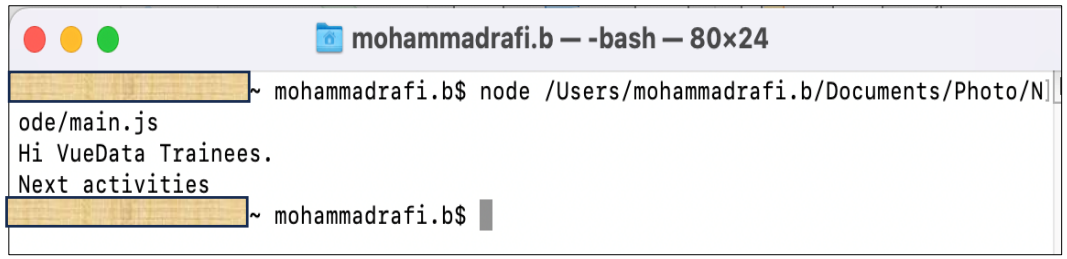
- Entry Point (app.js or index.js)
- Import required modules
- Routing (incoming requests handling)
- Middleware (Authentication and Authorization, Logging or Validations)
- Database Connectivity
- Templates Engine (Optional)
- Error Handling
- Security (input sanitization, XSS and SQL Injection protection
- Caching
- Internationalization and Localization
- Session Management

Node JS – Running Server

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

http.createServer(function (request, response) {
   // Send the HTTP header
   // HTTP Status: 200 : OK
   // Content Type: text/plain
   response.writeHead(200, {'Content-Type': 'text/plain'});

   // Send the response body as "Hello World"
   response.end('Hello VueData\n');
}).listen(8081);

// Console will print the message
console.log('Server running at http://127.0.0.1:8081/');
```

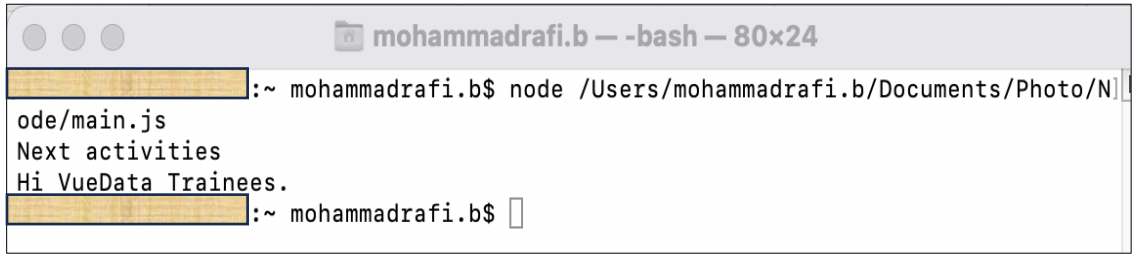
Node JS – Blocking and Non-Blocking

```
var fs = require("fs");
var data = fs.readFileSync('/Users/mohammadrafi.b/Documents/Photo/Node/text.txt');

console.log(data.toString());
console.log("Next activities");
```



BLOCKING

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

fs.readFile('/Users/mohammadrafi.b/Documents/Photo/Node/text.txt', function (err, data) {
   if (err) return console.error(err);
   console.log(data.toString());
});

console.log("Next activities");
```



NON - BLOCKING

Node JS – Modules

- **What are Modules?**
  - Organized reusable code written for specific purpose.

- **What are the types of Modules?**
  - **Core Modules**
    - **fs (File Stream)**
      - `const fs = require('fs');`
    - **http**
      - `const http = require('http');`
    - **https**
      - `const https = require('https');`
    - **path**
      - `const path = require('path');`
    - **os (Operating System)**
      - `const os = require('os');`
    - **Events**
      - `const EventEmitter = require('events');`
    - **Util**
      - `const util = require('util');`
    - **Querystring**
      - `const querystring = require('querystring');`
    - **url**
      - `const url = require('url');`

User- Defined Modules

```js
// MathOperations.js

// Function to add two numbers
const add = (a, b) => {
    return a + b;
};

// Function to subtract two numbers
const subtract = (a, b) => {
    return a - b;
};

// Function to multiply two numbers
const multiply = (a, b) => {
    return a * b;
};

// Function to divide two numbers
const divide = (a, b) => {
    if (b === 0) {
        throw new Error("Division by zero is not allowed");
    }
    return a / b;
};

// Export the functions
module.exports = {
    add,
    subtract,
    multiply,
    divide,
};
```

```js
// Main.js

// Import the MathOperations module
const math = require('./MathOperations');

// Use the functions from the MathOperations module
const resultAdd = math.add(5, 3);
console.log('5 + 3 =', resultAdd);

const resultSubtract = math.subtract(10, 4);
console.log('10 - 4 =', resultSubtract);

const resultMultiply = math.multiply(6, 7);
console.log('6 * 7 =', resultMultiply);

try {
    const resultDivide = math.divide(8, 0);
    console.log('8 / 0 =', resultDivide);
} catch (error) {
    console.error('Error:', error.message);
}
```

Node JS – Routing

- **What is Routing?**
  - process of directing incoming HTTP requests to specific handlers or controllers based on the requested URL and HTTP method

- **What are the types of Routing?**
  - **Basic (URL Parsing) Routing**
  - **Express JS Routing**
  - **RESTful Routing**
  - **Middleware Routing**
  - **Modular Routing**
  - **Controller-Based Routing**

```
// Create an HTTP server
const server = http.createServer((req, res) => {
    // Parse the requested URL
    const url = new URL(req.url, 'http://localhost:3000');

    // Routing based on the requested path
    if (url.pathname === '/') {
        // Handle the root (home) route
        res.writeHead(200, { 'Content-Type': 'text/plain' });
        res.end('Welcome to the home page!');
    } else if (url.pathname === '/about') {
        // Handle the about route
```

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

app.get('/', (req, res) => {
    res.send('Home Page');
});

app.get('/about', (req, res) => {
    res.send('About Us');
});
```

```
app.get('/api/users', (req, res) => {
    // Retrieve a list of users
});

app.post('/api/users', (req, res) => {
    // Create a new user
});

app.put('/api/users/:id', (req, res) => {
    // Update user with the specified ID
});

app.delete('/api/users/:id', (req, res) => {
    // Delete user with the specified ID
});
```

```
app.use('/admin', (req, res, next) => {
    // Middleware for admin routes
});

app.use('/api', (req, res, next) => {
    // Middleware for API routes
});
```

```
const express = require('express');
const router = express.Router();

router.get('/products', (req, res) => {
    // Handle product listing
});

router.get('/products/:id', (req, res) => {
    // Handle product details
});

module.exports = router;
```

```
// UserController.js
module.exports = {
    getUser: (req, res) => {
        // Retrieve user details
    },
    updateUser: (req, res) => {
        // Update user information
    },
};

// Routes.js
const express = require('express');
const router = express.Router();
const UserController = require('./UserController');

router.get('/users/:id', UserController.getUser);
router.put('/users/:id', UserController.updateUser);

module.exports = router;
```

Node JS – Middleware

- **What is Middleware?**
  - **allows you to perform tasks and processing in the request-response cycle before it reaches the final route handler**

- **What are the types of Middleware?**
  - **Application-Level Middleware (**Logging middleware, body parsing middleware, authentication middleware.**)**
  - **Route-Specific Middleware (**Authentication middleware for specific routes, route-specific logging middleware**)**
  - **Error-Handling Middleware (**Error logging middleware, custom error-handling middleware**)**
  - **Third-Party Middleware (**Express.js middleware like express-session, passport for authentication**)**
  - **Built-In Middleware (**express.json() for parsing JSON data, express.static() for serving static files.**)**
  - **Custom Middleware (**Custom authentication middleware, middleware for request validation**)**
  - **Logging Middleware (**logs information about incoming requests, such as request method, URL, and timestamp**)**
  - **Authentication Middleware (**verifies the identity of users or clients before allowing access to certain routes or resources**)**
  - **Security Middleware (**adds security features to an application, such as preventing cross-site scripting (XSS) attacks or enforcing HTTPS**)**
  - **CORS Middleware (**handling Cross-Origin Resource Sharing (CORS) to control which domains are allowed to access resources on a web server**)**
  - **Session Middleware (**manages user sessions, often used in web applications to maintain user state between requests**)**
  - **Compression Middleware (**compressing server responses to reduce data transfer size and improve performance**)**
  - **Request Validation Middleware (**validates incoming request data to ensure it meets specific criteria or constraints**)**

Node JS – Event Emitter

```
const EventEmitter = require('events');

class ChatRoom extends EventEmitter {
    sendMessage(user, message) {
        // Process the message
        console.log(`${user}: ${message}`);

        // Emit a "new message" event
        this.emit('newMessage', { user, message });
    }
}

const chatRoom = new ChatRoom();

// Listen for "new message" events
chatRoom.on('newMessage', ({ user, message }) => {
    console.log(`Received message from ${user}: ${message}`);
});

// Send a message
chatRoom.sendMessage('UserA', 'Hello, everyone!');
```

```
[redacted] mohammadrafi.b$ node /Users/mohammadrafi.b/Documents/Photo/N
ode/main.js
UserA: Hello, everyone!
Received message from UserA: Hello, everyone!
[redacted] mohammadrafi.b$
```

- addListener(event, listener)
- on(event, listener)
- once(event, listener)
- removeListener(event, listener)
- removeAllListeners([event])
- setMaxListeners(n)
- listeners(event)
- emit(event, [arg1], [arg2], [...])

Node JS – Streams and Pipes

- **What are Streams?**
  - **An object that lets us to read/write/transform data from source to destination.**

- **What are the types of Streams?**

Readable	Writeable	Duplex	Tansform

- **What are some common events for Streams?**

Data	End	Error	Finish

- **What is Pipe?**
  - a mechanism where we provide the output of one stream as the input to another stream
- **What is Pipe Chaining?**
  - a mechanism to connect the output of one stream to another stream and create a chain of multiple stream operations.

Node JS – Streams and Pipes

```
const fs = require('fs');
const { Transform } = require('stream');
// Create a readable stream from the input file
const readStream = fs.createReadStream('/Users/mohammadrafi.b/Documents/Photo/Node/text.txt', 'utf8');
// Create a writable stream to the output file
const writeStream = fs.createWriteStream('/Users/mohammadrafi.b/Documents/Photo/Node/capitaltext.txt', 'utf8');
// Create a transform stream to capitalize each line
const capitalizeStream = new Transform({
    transform(chunk, encoding, callback) {
        const capitalizedChunk = chunk.toString().toUpperCase();
        this.push(capitalizedChunk);
        callback();
    }
});
// Pipe the data through the streams
readStream.pipe(capitalizeStream).pipe(writeStream);

// Event handling
readStream.on('data', (chunk) => {
    console.log('Data chunk received:', chunk);
});
readStream.on('end', () => {
    console.log('Read stream ended');
});
readStream.on('error', (err) => {
    console.error('Read stream error:', err);
});
writeStream.on('finish', () => {
    console.log('Write stream finished writing to capitaltext.txt');
});
writeStream.on('error', (err) => {
    console.error('Write stream error:', err);
});
```

```
[redacted]~ mohammadrafi.b$ node /Users/mohammadrafi.b/Documents/Photo/N
ode/main.js
Data chunk received: Hi VueData Trainees.
Read stream ended
Write stream finished writing to capitaltext.txt
```

capitaltext.txt

```
HI VUEDATA TRAINEES.
```

Node JS – Web Server and Web Client

```
const http = require('http');
const fs = require('fs');

const server = http.createServer((req, res) => {
    // Handle HTTP GET requests
    if (req.method === 'GET') {
        if (req.url === '/') {
            // Serve an HTML page to the client
            fs.readFile(
                '/Users/mohammadrafi.b/Documents/Photo/Node/text.txt', 'utf8',
                (err, data) => {
                if (err) {
                    res.writeHead(500, { 'Content-Type': 'text/plain' });
                    res.end('Internal Server Error');
                } else {
                    res.writeHead(200, { 'Content-Type': 'text/plain' });
                    res.end(data);
                }
            });
        } else if (req.url === '/api/data') {
            // Respond with JSON data
            const responseData = { message: 'Hello, client!' };
            res.writeHead(200, { 'Content-Type': 'application/json' });
            res.end(JSON.stringify(responseData));
        } else {
            // Handle other routes
            res.writeHead(404, { 'Content-Type': 'text/plain' });
            res.end('Not Found');
        }
    }
});

const port = 3000;
server.listen(port, () => {
    console.log(`Server is listening on port ${port}`);
});
```

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

const options = {
    hostname: 'localhost',
    port: 3000,
    path: '/api/data',
    method: 'GET',
};

const req = http.request(options, (res) => {
    let data = '';

    res.on('data', (chunk) => {
        data += chunk;
    });

    res.on('end', () => {
        const responseData = JSON.parse(data);
        console.log(`Received data from server: ${JSON.stringify(responseData)}`);
    });
});

req.on('error', (error) => {
    console.error(`Request error: ${error.message}`);
});

req.end();
```

Node JS – Database Connectivity

```
const mysql = require('mysql2');

// Create a connection pool
const pool = mysql.createPool({
  host: 'localhost', // Replace with your MySQL server host
  user: 'your_username', // Replace with your MySQL username
  password: 'your_password', // Replace with your MySQL password
  database: 'your_database_name', // Replace with your MySQL database name
  waitForConnections: true,
  connectionLimit: 10, // Adjust the connection pool size as needed
  queueLimit: 0,
});

// Execute a SELECT query
pool.query('SELECT * FROM your_table_name', (err, results, fields) => {
    if (err) {
      console.error('Error executing query:', err);
      return;
    }

    // Process the query results (results contains the rows returned by the query)
    console.log('Query results:', results);

    // You can also access metadata about the result set via the fields parameter
    console.log('Query fields:', fields);
  });

  // Close the connection pool when you're done
pool.end((err) => {
    if (err) {
      console.error('Error closing pool:', err);
    }
    console.log('Connection pool closed');
  });
```

```
const { MongoClient } = require('mongodb');

const url = 'mongodb://localhost:27017'; // Replace with your M
const dbName = 'AddressBook';

async function connectAndQuery() {
  const client = new MongoClient(url, {
    useNewUrlParser: true,
    useUnifiedTopology: true,
  });

  try {
    // Connect to the MongoDB server
    await client.connect();
    console.log('Connected to MongoDB');

    const db = client.db(dbName);
    const collection = db.collection('Products');

    // Perform a sample query (find all documents)
    const docs = await collection.find({}).toArray();
    console.log('Documents:', docs);
  } catch (error) {
    console.error('Error:', error);
  } finally {
    // Close the MongoDB connection
    await client.close();
    console.log('Disconnected from MongoDB');
  }
}

connectAndQuery();
```

Node JS – Destructing Arrays

```
// Basic array destructuring
const [first, second, third] = [1, 2, 3];
console.log(first);  // 1
console.log(second); // 2
console.log(third);  // 3

// Skipping elements
const [one, , three] = [1, 2, 3];
console.log(one);   // 1
console.log(three); // 3

// Rest operator (collects the remaining elements into an array)
const [a, ...rest] = [1, 2, 3, 4, 5];
console.log(a);    // 1
console.log(rest); // [2, 3, 4, 5]
```

Node JS – Destructing Arrays

```
// Basic array destructuring
const [first, second, third] = [1, 2, 3];
console.log(first);  // 1
console.log(second); // 2
console.log(third);  // 3

// Skipping elements
const [one, , three] = [1, 2, 3];
console.log(one);   // 1
console.log(three); // 3

// Rest operator (collects the remaining elements into an array)
const [a, ...rest] = [1, 2, 3, 4, 5];
console.log(a);    // 1
console.log(rest); // [2, 3, 4, 5]
```

Node JS – Destructing Objects

```
// Basic object destructuring
const person = { firstName: 'John', lastName: 'Doe' };
const { firstName, lastName } = person;
console.log(firstName); // 'John'
console.log(lastName);  // 'Doe'

// Renaming variables while destructuring
const { firstName: first, lastName: last } = person;
console.log(first); // 'John'
console.log(last);  // 'Doe'

// Default values
const { age = 30 } = person; // If 'age' is not defined in 'person', it defaults to 30
console.log(age); // 30

// Nested object destructuring
const user = {
  fname: 'Alice',
  address: {
    street: '123 Main St',
    city: 'Wonderland'
  }
};
const { fname, address: { city } } = user;
console.log(fname); // 'Alice'
console.log(city); // 'Wonderland'
```

Node JS – Function Param, Arrays & Objects

```
// Destructuring in function parameters
function printPerson({ firstName, lastName }) {
    console.log(`${firstName} ${lastName}`);
}

const person = { firstName: 'Jane', lastName: 'Doe' };
printPerson(person); // 'Jane Doe'
```

```
// Destructuring return values of a function
function getNumbers() {
    return [1, 2, 3];
}

const [x, y, z] = getNumbers();
console.log(x); // 1
console.log(y); // 2
console.log(z); // 3
```

```
// Destructuring return values of a function returning an object
function getUser() {
    return { name: 'Bob', age: 25 };
}

const { name, age } = getUser();
console.log(name); // 'Bob'
console.log(age);  // 25
```

PROJECT TRAINING WORKSHOP

Express JS

Express JS

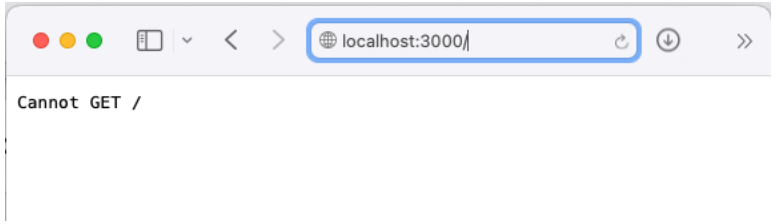
- **What is Express JS?**
  - minimal and flexible Node.js web application framework that provides a set of essential features and tools for building web and API applications

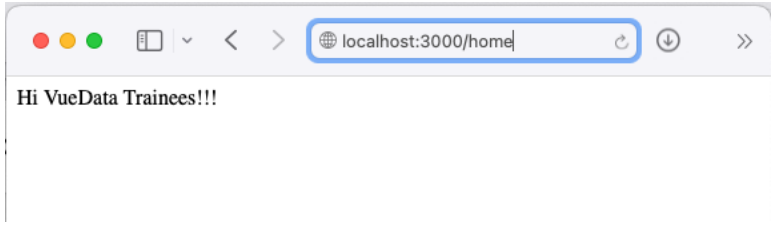
- **What are the roles of Express JS?**

Routing (routing HTTP methods)	Middleware (logging, authentication, authorization, data validation)	HTTP Utility Methods (parsing request body, cookie and sessions, headers)	Template Engines (rendering dynamic web pages)
Static File Serving (express.static middleware)	Error Handling	Integration with Database	API Development (RESTFul APIs)

Express JS – Running Application

```
var express = require('express');
var app = express();

app.get('/home', function(req, res){
   res.send("Hi VueData Trainees!!!");
});

app.listen(3000);
```





Express JS – Routing & HTTP Methods

```
var express = require('express');
var app = express();

app.get('/home', function(req, res){
   res.send("Hi VueData Trainees!!!");
});

app.listen(3000);
```

```
//employeeroute.js

var express = require('express');
var router = express.Router();

router.get('/', function(req, res){
   res.send('Employee GET route called.');
});
router.post('/', function(req, res){
   res.send('Employee POST route called.');
});

//export this router to use in our index.js
module.exports = router;
```

```
//main.js

var express = require('Express');
var app = express();

var employee = require('./employeeroute.js');

//both index.js and things.js should be in same directory
app.use('/employee', employee);

app.listen(3000);
```

HTTP Methods

app.method(path, handler)
app.all(path, handler)

Express JS – URL Binding

Basic

```
app.get('/', (req, res) => {
    res.send('Hello, Express!');
});
```

Dynamic

```
app.get('/users/:id', (req, res) => {
    const userId = req.params.id;
    res.send(`User ID: ${userId}`);
});
```

Optional

```
app.get('/profile/:name?', (req, res) => {
    const name = req.params.name || 'Guest';
    res.send(`Profile: ${name}`);
});
```

Wildcard

```
app.get('/articles/*', (req, res) => {
    res.send('Wildcard Route: Articles');
});
```

Regular Expression

```
app.get(/\/[0-9]{3}-[0-9]{2}-[0-9]{4}/, (req, res) => {
    res.send('Regex Route: SSN');
});
```

Query Parameters

```
app.get('/search', (req, res) => {
    const query = req.query.q;
    res.send(`Search query: ${query}`);
});
```

Route with Multiple Handlers

```
const authMiddleware = (req, res, next) => {
    // Check authentication
    if (req.isAuthenticated()) {
        next();
    } else {
        res.status(401).send('Unauthorized');
    }
};

app.get('/secure', authMiddleware, (req, res) => {
    res.send('Authenticated Route');
});
```

Express JS – Logger Middleware

```
const express = require('express');
const app = express();
const port = 3000;

// Logger Middleware
const loggerMiddleware = (req, res, next) => {
  console.log(`[${new Date()}] ${req.method} ${req.url}`);
  next();
};

// Register the Logger Middleware
app.use(loggerMiddleware);

// Route
app.get('/', (req, res) => {
  res.send('Hello, World!');
});

// Start the Express.js server
app.listen(port, () => {
  console.log(`Server is running on port ${port}`);
});
```

Express JS – Body Parser Middleware

```
const express = require('express');
const bodyParser = require('body-parser'); // Import body-parser
const app = express();
const port = 3000;

// Use body-parser middleware to parse JSON and URL-encoded request bodies
app.use(bodyParser.json()); // Parse JSON bodies
app.use(bodyParser.urlencoded({ extended: true })); // Parse URL-encoded bodies

// Define a POST route that expects a JSON request body
/* {
    "name": "John",
    "age": 30,
    "city": "New York"
}
Set Content-Type to application/application/json
*/
app.post('/json', (req, res) => {
  const requestData = req.body; // Access the parsed JSON body
  res.json({ message: 'Received JSON data', data: requestData });
});

// Define a POST route that expects URL-encoded form data
// name=Alice&email=alice@example.com&city=San+Francisco , Set Content-Type to application/x-www-form-urlencoded
app.post('/urlencoded', (req, res) => {
  const formData = req.body; // Access the parsed URL-encoded body
  res.json({ message: 'Received URL-encoded form data', data: formData });
});

// Start the Express.js server
app.listen(port, () => {
  console.log(`Server is running on port ${port}`);
});
```

Express JS – Authentication Middleware

```
const express = require('express');
const app = express();
const port = 3000;

// Simulated User Database
const users = [
  { username: 'user1', password: 'password1' },
  { username: 'user2', password: 'password2' },
];

// Authentication Middleware
const authenticateMiddleware = (req, res, next) => {
  // Simulate user authentication (you can replace this with actual authentication logic)
  const { username, password } = req.query;
  const authenticatedUser = users.find((user) => user.username === username && user.password === password);

  if (authenticatedUser) {
    // User is authenticated, proceed to the next middleware or route handler
    next();
  } else {
    // User is not authenticated, send a 401 Unauthorized response
    res.status(401).send('Unauthorized');
  }
};

// Register the Authentication Middleware for a Protected Route
app.get('/protected', authenticateMiddleware, (req, res) => {
  res.send('Protected Route: You are authenticated!');
});

// Public Route (No Authentication Required)
app.get('/public', (req, res) => {
  res.send('Public Route: No authentication required.');
});

// Start the Express.js server
app.listen(port, () => {
  console.log(`Server is running on port ${port}`);
});
```

http://localhost:3000/public

http://localhost:3000/protected

http://localhost:3000/protected?username=user1
&password=password1

Express JS – Template Engines (PUG/JADE)

```
const express = require('express');
const app = express();
const port = 3000;

app.set('view engine', 'pug'); // Set Pug as the view engine
app.set('views', '/Users/mohammadrafi.b/Documents/Photo/Node/PugSample/views');

app.use(express.static('/Users/mohammadrafi.b/Documents/Photo/Node/PugSample/public'));


// Define a route that renders a Pug template
app.get('/example', (req, res) => {
  res.render('example', { title: 'Express with Pug', message: 'Hello, Pug!' });
});

// Start the Express.js server
app.listen(port, () => {
  console.log(`Server is running on port ${port}`);
});
```

```
//inline CSS
doctype html
html
  head
    title= title
  body
    - const textColor = 'blue';
    - const bgColor = 'yellow';
    h1(style=`color: ${textColor};`)= message
    div(style=`background-color: ${bgColor}; padding: 10px;`)= message + ' your message added to DIV'
```

```
//External CSS
doctype html
html
  head
    title= title
    link(rel='stylesheet', href='/styles/style.css')
  body
    h1(class='highlighted-text')= message
    div(class='highlighted-text')= message + ' your message added to DIV'
```

Express JS – Template Engines (Handlebars)

```
var express = require("express");
var exphbs = require("express-handlebars");
var app = express();
var port = 5000;

// Handlebars
var hbs = exphbs.create();
app.engine("handlebars", hbs.engine);
app.set("view engine", "handlebars");
app.use(express.static(__dirname + "/views"));

app.get("/", function(req, res) {
  res.render("home", {
    title: "Home Page",
    content: "This is an example of Express with Handlebars"
  });
});

app.get("/about", function(req, res) {
    res.render("about", {
      title: "About Us",
      content: "This is an example of Express with Handlebars"
    });
  });

var server = app.listen(port, function() {
  console.log("Listening on port %d", server.address().port);
});
```

layouts > main.handlebars > html

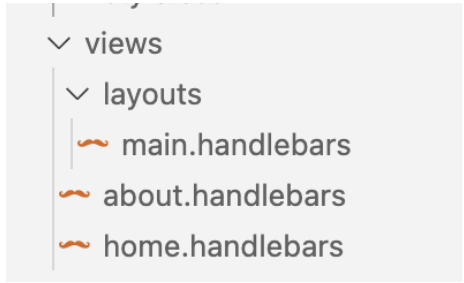
```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <title></title>
  <meta name="viewport" content="width=device-width, initial-scale=1.0,
  maximum-scale=1.0" />

  <link href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css"
  rel="stylesheet" />
  <style>
    @media only screen and (max-width : 750px) {}
  </style>
</head>
<body>
  {{{body}}}
```

about.handlebars > div.container.mt-5

```
<div class="container mt-5">
  <h2>{{title}}</h2>
  <p>{{content}}</p>
```

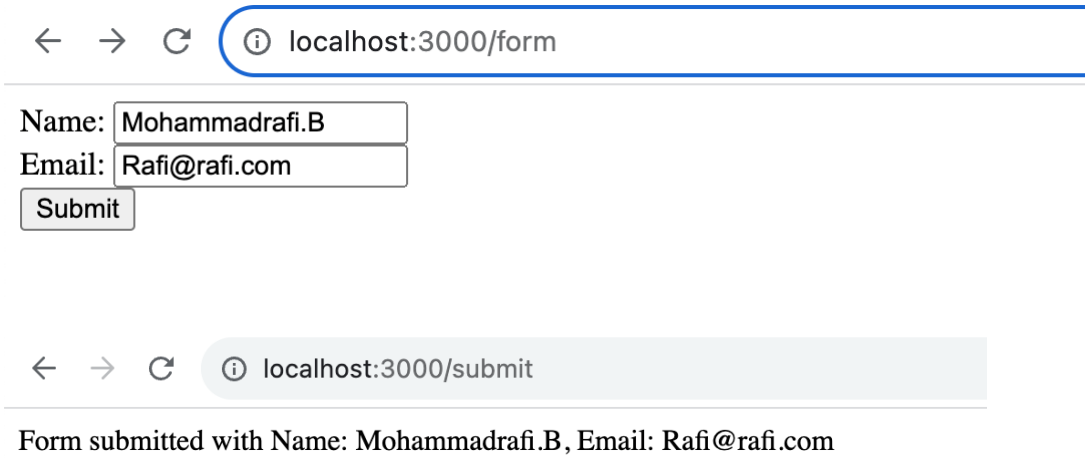
home.handlebars > div.container.mt-5

```
<div class="container mt-5">
  <h2>{{title}}</h2>
  <p>{{content}}</p>
</div>
```



Express JS – Form Data Submission

```
const express = require('express');
const bodyParser = require('body-parser');
const app = express();
const port = process.env.PORT || 3000;

// Use body-parser middleware to parse JSON and URL-encoded form data
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: true }));

app.set('view engine', 'pug'); // Set Pug as the view engine

// Define a route to serve an HTML form
app.get('/form', (req, res) => {
    res.render('form');
  });

// Define a route to handle the form submission
app.post('/submit', (req, res) => {
  const name = req.body.name;
  const email = req.body.email;
  res.send(`Form submitted with Name: ${name}, Email: ${email}`);
});

// Start the server
app.listen(port, () => {
  console.log(`Server is running on port ${port}`);
});
```



Express JS – RESTFul Methods

```
const express = require('express');
const bodyParser = require('body-parser');
const app = express();
// Sample data (in-memory database)
let items = [
  { id: 1, name: 'Item 1' }, { id: 2, name: 'Item 2' },
  { id: 3, name: 'Item 3' },
];

app.use(bodyParser.json());

// GET all items
app.get('/items', (req, res) => {
  res.json(items);
});

// GET a single item by ID
app.get('/items/:id', (req, res) => {
  const id = parseInt(req.params.id);
  const item = items.find((item) => item.id === id);
  if (!item) { res.status(404).json({ message: 'Item not found' }
  } else { res.json(item); }
});

// POST (create) a new item
app.post('/items', (req, res) => {
  const newItem = req.body;
  items.push(newItem);
  res.status(201).json(newItem);
});
```

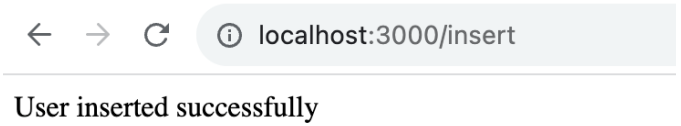
```
app.put('/items/:id', (req, res) => {
  const id = parseInt(req.params.id);
  const updatedItem = req.body;
  const index = items.findIndex((item) => item.id === id);

  if (index === -1) {
    res.status(404).json({ message: 'Item not found' });
  } else {
    items[index] = { id, ...updatedItem };
    res.json(items[index]);
  }
});

// DELETE an item by ID
app.delete('/items/:id', (req, res) => {
  const id = parseInt(req.params.id);
  const index = items.findIndex((item) => item.id === id);

  if (index === -1) {
    res.status(404).json({ message: 'Item not found' });
  } else {
    const deletedItem = items.splice(index, 1);
    res.json(deletedItem[0]);
  }
});

// Start the server
app.listen(3000, () => {
  console.log(`Server is running on port ${port}`);
});
```

RAAFI
Builds You Best

Express JS – Database

```
//user.js

const mongoose = require('mongoose');

const userSchema = new mongoose.Schema({
  name: String,
  email: String,
});

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

module.exports = User;
```

```
const express = require('express');
const mongoose = require('mongoose');
const User = require('./Model/user'); // Import the User schema
const app = express();
const port = process.env.PORT || 3000;

// Connect to MongoDB
mongoose.connect('mongodb://localhost/mydatabase', {
  useNewUrlParser: true,
  useUnifiedTopology: true,
});

const db = mongoose.connection;
db.on('error', console.error.bind(console, 'MongoDB connection error:'));
db.once('open', () => {
  console.log('Connected to MongoDB');
});

// Define a route to insert a new user
app.get('/insert', async (req, res) => {
  const newUser = new User({
    name: 'John Doe',
    email: 'john@example.com',
  });

  try {
    await newUser.save();
    res.send('User inserted successfully');
  } catch (error) {
    console.error(error);
    res.status(500).send('Error inserting user');
  }
});

// Start the server
app.listen(port, () => {
  console.log(`Server is running on port ${port}`);
});
```



Express JS – Assignment

Create Express JS (Handlebars or Pug) application for Meeting room booking application using Mongo DB

PROJECT TRAINING WORKSHOP

TypeScript JS

TypeScript JS

- **What is TypeScript JS?**
  - **Making JavaScript code more type-safety**
- **What are the roles of TypeScript JS?**

Static Typing	Code Readability	Better Refactoring	Safety and Robustness
Enhanced Tooling (autocomplete, type checking, code navigation)	Compatibility	Ecosystem and Community	Large Codebase Management

TypeScript JS

- **What are the key concepts of TypeScript JS?**

Static Typing	Type Annotations	Interfaces	Type Inference
Union Types (number | string)	Enums	Classes & Access Modifiers	Generics
Modules	Type Assertion	Declaration Files (.d.ts)	Utility Types

TypeScript JS – Static Typing and Type Annotations & Tuples

```ts
let age: number = 30;
let name: string = 'John';

function add(a: number, b: number): number {
  return a + b;
}
```

Special Types

```ts
let x: any = true;

let x: unknown = 1;

let x: never = true;

let y: undefined = undefined;
let z: null = null;
```

Tuples

```ts
let values: [number, boolean, string]; values = [5, false, 'Tuple string'];

const values: readonly [number, boolean, string] = [5, true, 'Tuple string'];

const graph: [x: number, y: number] = [55.2, 41.3];
```

TypeScript JS – Type Annotations

Variable Type

```
// Explicitly annotate the type of a variable
let age: number = 30;
let name: string = "Alice";
let isStudent: boolean = true;
```

Function Parameter and Return Type

```
// Annotate the parameter and return types of a function
function add(x: number, y: number): number {
    return x + y;
}

function greet(name: string): string {
    return `Hello, ${name}!`;
}
```

Array and Object Type

```
// Annotate the types of elements in an array
let numbers: number[] = [1, 2, 3];

// Annotate the structure of an object
let person: { name: string; age: number } = { name: "Bob", age: 25 };
```

Custom Type

```
// Create a type alias
type Point = { x: number; y: number };

// Annotate variables with the custom type
let p1: Point = { x: 1, y: 2 };
let p2: Point = { x: 3, y: 4 };
```

Union Type

```
// Annotate a variable with a union type
let result: number | string = Math.random() > 0.5 ? "error" : 42;
```

Function Type

```
// Annotate the type of a function itself
let calculator: (x: number, y: number) => number = (x, y) => x + y;
```

Tuple Type

```
// Annotate a tuple type
let pair: [string, number] = ["apple", 5];
```

TypeScript JS – Interfaces & Type Inference

```typescript
interface Person {
  name: string;
  age: number;
}

function greet(person: Person) {
  console.log(`Hello, ${person.name}`);
}

const john: Person = { name: 'John', age: 25 };
greet(john);
```

```typescript
interface Vehicle {
    wheels: number;
}

interface Car extends Vehicle {
    make: string;
    model: string;
}

// Usage
const myCar: Car = {
    wheels: 4,
    make: "Honda",
    model: "Civic",
};
```

```typescript
interface Shape {
    area(): number;
}

class Circle implements Shape {
    constructor(private radius: number) {}

    area(): number {
        return Math.PI * this.radius * this.radius;
    }
}

// Usage
const circle = new Circle(5);
console.log(circle.area()); // Output: 78.54
```

TypeScript JS – Type Inference

Basic Type

```
let age = 25; // Inferred type: number
let name = "Alice"; // Inferred type: string
let isAdmin = false; // Inferred type: boolean
```

Array Type

```
let numbers = [1, 2, 3]; // Inferred type: number[]
let names = ["Alice", "Bob", "Charlie"]; // Inferred type: string[]
```

Union Type

```
let value = Math.random() > 0.5 ? "error" : 42; // Inferred type: string
```

Function Type

```
function add(x, y) {
    return x + y;
}

// Inferred types: (x: any, y: any) => any
```

Object Type

```
let person = {
    firstName: "John",
    lastName: "Doe",
};

// Inferred type: { firstName: string, lastName: string }
```

Type Inference Type

```
let message = "This is a string";
let strLength = (message as string).length; // Override inference to string
```

TypeScript JS – Union Types & Enums

```
let value: number | string;
value = 42;    // Valid
value = 'hi';  // Valid
```

```
enum Color {
  Red,
  Green,
  Blue,
}

let color: Color = Color.Red; // Represents 0
```

TypeScript JS – Classes, Access Modifiers

```
class Animal {
  constructor(public name: string) {}
  public speak() {
    console.log(`The ${this.name} speaks.`);
  }
}

const dog = new Animal('Dog');
dog.speak();
```

TypeScript JS – Generics

```ts
function customSort<T>(arr: T[]): T[] {
    return arr.sort();
}

// Usage
const numbers = [3, 1, 2];
const sortedNumbers = customSort(numbers); // [1, 2, 3]

const words = ["apple", "cherry", "banana"];
const sortedWords = customSort(words); // ["apple", "banana", "cherry"]
```

```ts
function promiseWrapper<T>(value: T): Promise<T> {
    return Promise.resolve(value);
}

// Usage
const numberPromise = promiseWrapper(42);
numberPromise.then((result) => console.log(result)); // 42

const stringPromise = promiseWrapper("Hello");
stringPromise.then((result) => console.log(result)); // "Hello"
```

```ts
async function fetchJson<T>(url: string): Promise<T> {
    const response = await fetch(url);
    return response.json();
}

// Usage
const userData = await fetchJson<User>("https://api.example.com/user/1");
const postList = await fetchJson<Post[]>("https://api.example.com/posts");
```

```ts
interface FormField<T> {
    value: T;
    required: boolean;
}

function validateField<T>(field: FormField<T>): boolean {
    if (field.required && !field.value) {
        return false; // Field is required but empty
    }
    return true;
}

// Usage
const usernameField: FormField<string> = { value: "john_doe", required: true };
const isValid = validateField(usernameField);
```

```ts
interface ButtonProps<T> {
    label: string;
    onClick: (value: T) => void;
}

function Button<T>(props: ButtonProps<T>) {
    return <button onClick={() => props.onClick(props.value)}>{props.label}</button>;
}

// Usage
<Button<number> label="Click Me" onClick={(value) => console.log(value)} value={42} />;
```

TypeScript JS – Modules

```ts
// math.ts
export function add(a: number, b: number): number {
  return a + b;
}

// main.ts
import { add } from './math';
console.log(add(5, 3)); // Outputs 8
```

```ts
// logger.ts
export default function log(message: string): void {
    console.log(message);
}

// main.ts
import logger from "./logger";
logger("Hello, world!"); // Logs "Hello, world!"
```

```ts
export class Circle {
    constructor(public radius: number) {}
    area(): number {
        return Math.PI * this.radius ** 2;
    }
}

// main.ts
import { Circle } from "./shapes";
const circle = new Circle(5);
console.log(circle.area()); // 78.53981633974483
```

```ts
// shapes.ts
export namespace Shapes {
    export class Circle {}
    export class Rectangle {}
}

// main.ts
import { Shapes } from "./shapes";
const circle = new Shapes.Circle();
```

TypeScript JS – Declaration Files

```
// myGlobal.d.ts
declare var myGlobal: string;

// main.ts
console.log(myGlobal); // TypeScript knows myGlobal exists
```

```
// math-library.d.ts
declare module "math-library" {
    export function add(a: number, b: number): number;
}

// main.ts
import { add } from "math-library";
console.log(add(5, 3)); // TypeScript recognizes the module's structure
```

TypeScript JS – Utility Types

Partial (T)

```ts
interface Person {
  name: string;
  age: number;
}

type PartialPerson = Partial<Person>;
// PartialPerson is equivalent to { name?: string; age?: number; }
```

Required<T>

```ts
interface OptionalPerson {
  name?: string;
  age?: number;
}

type RequiredPerson = Required<OptionalPerson>;
// RequiredPerson is equivalent to { name: string; age: number; }
```

Readonly<T>

```ts
interface Config {
  apiKey: string;
  endpoint: string;
}

type ReadonlyConfig = Readonly<Config>;
// ReadonlyConfig has the same structure as Config but with readonly
```

Record<K, T>

```ts
type PhoneBook = Record<string, string>;
// PhoneBook is an object with string keys and string values
```

Pick<K, T>

```ts
interface User {
  id: number;
  name: string;
  email: string;
}

type UserInfo = Pick<User, 'name' | 'email'>;
// UserInfo is equivalent to { name: string; email: string; }
```

Omit<K, T>

```ts
interface Car {
  make: string;
  model: string;
  year: number;
}

type CarSpecs = Omit<Car, 'year'>;
// CarSpecs is equivalent to { make: string; model: string; }
```

Exclude<K,U>

```ts
type Numbers = 1 | 2 | 3 | 4 | 5;
type OddNumbers = Exclude<Numbers, 2 | 4>;
// OddNumbers is equivalent to 1 | 3 | 5
```

Extract<K,U>

```ts
type Fruit = 'apple' | 'banana' | 'orange' | 'grape';
type Citrus = Extract<Fruit, 'orange' | 'grape'>;
// Citrus is equivalent to 'orange' | 'grape'
```

ReturnType<T>

```ts
function greet(name: string): string {
  return `Hello, ${name}!`;
}

type GreetReturnType = ReturnType<typeof greet>;
// GreetReturnType is equivalent to string
```

Parameters<T>

```ts
function calculateSum(a: number, b: number): number {
  return a + b;
}

type SumParams = Parameters<typeof calculateSum>;
// SumParams is equivalent to [number, number]
```