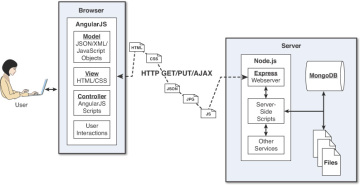
**P.V.P Siddhartha Institute of Technology**

**Department of Computer Science and Engineering**

**Subject Code: 20CS3603, Subject Name: MERN Stack Development:PVP20**

**Descriptive Examination -1**

1. A) Explain about the Node.js-to-Angular stack in detail. CO1, L2, 3M



The **Node.js-to-Angular stack** is a widely used full-stack JavaScript architecture that enables the development of dynamic and scalable web applications using JavaScript from end to end. At the frontend, **Angular** is a powerful TypeScript-based framework developed by Google that allows developers to build rich, responsive single-page applications (SPAs). Angular provides built-in support for routing, form handling, two-way data binding, and HTTP client modules, which makes it easy to interact with backend APIs and display dynamic content on the web page.

On the server side, **Node.js** serves as the runtime environment that executes JavaScript code outside of a browser. It is known for its non-blocking, event-driven architecture, making it highly suitable for building real-time and scalable server-side applications. Node.js works in conjunction with **Express.js**, a minimal and flexible web application framework that simplifies the creation of APIs, handles routing, and manages middleware for processing requests. Express acts as a bridge between the Angular frontend and the database.

For data storage, this stack typically uses either **MongoDB**, a NoSQL document-based database, or relational databases like **MySQL** or **PostgreSQL**. The backend (Node.js with Express) interacts with the database to perform CRUD operations and returns the results to the Angular frontend through RESTful APIs, usually in JSON format. This seamless integration allows developers to use JavaScript consistently across the entire application stack.

The Node.js-to-Angular stack is ideal for building modern web applications such as dashboards, content management systems, chat applications, and e-commerce platforms. It offers high performance, scalability, and a strong open-source community. Overall, this stack is a powerful solution for developers looking to build fast, interactive, and efficient web applications using JavaScript on both the client and server sides.

B) Build an HTML page including any required JavaScript that takes a number from

one text field in the range of 0 to 999 and shows it in another text field in words. If the

number is out of range, it should show “out of range” and if it is not a number, it

should show “not a number” message in the result box. CO1, L2, 2M

1. A) Demonstrate how to Publish, Install and functions in a Node.js Packaged Module from the NPM Registry CO2, L3, 5M

<!DOCTYPE html>

<html>

<head>

<title>Number to Words Converter</title>

<style>

body {

font-family: Arial;

padding: 20px;

}

input[type="text"] {

padding: 5px;

width: 200px;

}

#output {

font-weight: bold;

margin-top: 10px;

color: green;

}

</style>

</head>

<body>

<h2>Number to Words (0–999)</h2>

<label>Enter a number: </label>

<input type="text" id="numberInput" />

<button onclick="convertToWords()">Convert</button>

<div id="output"></div>

<script>

function convertToWords() {

const input = document.getElementById("numberInput").value.trim();

const output = document.getElementById("output");

if (isNaN(input) || input === "") {

output.textContent = "not a number";

output.style.color = "red";

return;

}

const num = parseInt(input);

if (num < 0 || num > 999) {

output.textContent = "out of range";

output.style.color = "orange";

return;

}

const ones = ["", "one", "two", "three", "four", "five", "six", "seven", "eight", "nine"];

const teens = ["ten", "eleven", "twelve", "thirteen", "fourteen", "fifteen",

"sixteen", "seventeen", "eighteen", "nineteen"];

const tens = ["", "", "twenty", "thirty", "forty", "fifty", "sixty", "seventy", "eighty", "ninety"];

function numberToWords(n) {

if (n === 0) return "zero";

let words = "";

if (Math.floor(n / 100) > 0) {

words += ones[Math.floor(n / 100)] + " hundred ";

n %= 100;

}

if (n >= 10 && n < 20) {

words += teens[n - 10];

} else {

if (Math.floor(n / 10) > 0) {

words += tens[Math.floor(n / 10)] + " ";

}

if (n % 10 > 0) {

words += ones[n % 10];

}

}

return words.trim();

}

output.textContent = numberToWords(num);

output.style.color = "green";

}

</script>

</body>

</html>

1. A) Explain Request, Response and server objects CO1, L2, 2M

**1. Request Object (req)**

The **Request object** represents the **incoming HTTP request** from the client (e.g., a browser, Postman, or mobile app) to the server.

**Request Method**: Type of request (GET, POST, PUT, DELETE)

**URL**: The endpoint/resource being requested

**Headers**: Metadata sent by the client (e.g., authentication tokens, content-type)

**Body**: Data sent by the client in POST/PUT requests (e.g., form data, JSON)

**Query Parameters**: Data sent via the URL (e.g., ?id=10)

**Cookies & Sessions**: Stored data about the client

**Example:**

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

const username = req.body.username; // Accessing form data

});

**2.. Response Object (req)**

The **Response object** represents the **HTTP response** that the server sends back to the client.

**Status Codes**: (e.g., 200 OK, 404 Not Found, 500 Server Error)

**Response Headers**: Metadata about the response

**Body/Content**: The actual data sent back (HTML, JSON, plain text)

**Redirects**: Can redirect client to another URL

**Cookies**: Can send cookies to the client

**Example:**

res.status(200).send('Login successful');

1. **Server Object**

The **Server object** represents the **web server itself**, which listens for incoming requests and routes them to the correct handler. It manages the **lifecycle of the application** and **network communication**.

Starts the application and listens on a port (e.g., 3000)

Routes requests to the appropriate functions

Manages connections and handles errors

**Example:**

const http = require('http');

const server = http.createServer((req, res) => {

res.write('Hello, World!');

res.end();

});

server.listen(3000, () => {

console.log('Server running on port 3000');

});

B) Implement HTTP Servers and Clients in Node.js with a suitable code example. CO2, L3,

2M

**HTTP Server in Node.js:**

The server listens on a port, accepts requests, and sends responses.

File: Server.js

const http = require('http');

// Create an HTTP server

const server = http.createServer((req, res) => {

// Set response header

res.writeHead(200, { 'Content-Type': 'text/plain' });

// Log the request URL and method

console.log(`Request received: ${req.method} ${req.url}`);

// Send a response

res.end('Hello from Node.js HTTP Server!');

});

// Start server on port 3000

server.listen(3000, () => {

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

});

**HTTP Client in Node.js:**

This client sends a request to the server and logs the response.

File: Client.js

const http = require('http');

// Set the request options

const options = {

hostname: 'localhost',

port: 3000,

path: '/',

method: 'GET'

};

// Make the HTTP request

const req = http.request(options, (res) => {

console.log(`Status Code: ${res.statusCode}`);

// Print response data

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

console.log(`Body: ${chunk.toString()}`);

});

});

// Handle any errors

req.on('error', (err) => {

console.error(`Error: ${err.message}`);

});

// End the request

req.end();

**Run commands:**

Start the server:

node server.js

Run the client in another terminal:

node client.js

will see the server log the request, and the client will print the server's response:

Status Code: 200

Body: Hello from Node.js HTTP Server!