Assignment-2

Subject: Web Development using MERN Stack **Paper Code : FSD-322T**

Q1. How does Node.js overcome the problem of blocking of I/O operations?

Node.js uses **non-blocking, event-driven architecture** with an **asynchronous I/O model**. It employs the **event loop** and **callbacks/promises** to handle multiple requests without waiting for I/O operations to complete.

```
const fs = require('fs');
fs.readFile('file.txt', 'utf8', (err, data) => {
   if (err) throw err;
   console.log(data); // Executes after reading file
});console.log("Reading file...");
```

Q2. Write down the features of Node JS. How HTTP requests and responses are handled by Express.js .

Features of Node.js:

- **Asynchronous & Non-blocking** – Handles multiple requests efficiently.
- **Single-threaded** – Uses an event loop for concurrency.
- **Fast Execution** – Built on Google V8 engine.
- **Scalable** – Handles many connections using event-driven architecture.
- **Rich Library Support** – Includes npm for packages.

Handling HTTP Requests & Responses in Express.js:

Express.js simplifies HTTP handling with middleware and routing.

Example:

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

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

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

Q3. Describe the concept of Middleware in Express.js by giving an example.

Middleware in Express.js

Middleware functions are functions that execute **before** the request reaches the route handler. They can modify req and res objects, execute code, or terminate requests.

Example: Logging Middleware

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

// Middleware function
app.use((req, res, next) => {
   console.log(`${req.method} request for ${req.url}`);
   next(); // Passes control to the next middleware/route
});

app.get('/', (req, res) => {
   res.send('Hello, Middleware!');
});
app.listen(3000, () => console.log('Server running on port 3000'));
```

Q4. Explain the Node.js event loop mechanism. How does it help in handling asynchronous operations?

Node.js Event Loop Mechanism

The **event loop** is the core of Node.js's **asynchronous** and **non-blocking** architecture. It continuously listens for events and executes callbacks in different phases.

Example:

```
console.log("Start");
setTimeout(() => console.log("Timeout"), 0);
setImmediate(() => console.log("Immediate"));
console.log("End");
```

How It Helps?

- **Non-blocking I/O**: Operations like file reading execute in the background.
- **Efficient Handling**: It prevents waiting for one request before processing another.
- **Scalability**: Handles multiple requests without needing multiple threads.

Q5. Explain in brief:

(a) Streams in Node.js

Streams allow efficient handling of large data by processing it in **chunks** instead of loading everything into memory.
Types: **Readable, Writable, Duplex, and Transform.**

Example:

```
const fs = require('fs');
const readStream = fs.createReadStream('file.txt');
readStream.on('data', chunk => console.log(chunk.toString()));
```

(b) Document in MongoDB

A **document** is a JSON-like structure used to store data in MongoDB.
Example:

```
{
 "_id": ObjectId("507f1f77bcf86cd799439011"),
 "name": "John",
 "age": 30
}
```

Each document is stored in a **collection**.

(c) Collection in MongoDB

A **collection** is a group of MongoDB documents (similar to a table in SQL).

Example: Collection named users

```
[
 { "name": "Alice", "age": 25 },
 { "name": "Bob", "age": 28 }
]
```

(d) Node.js Buffers

Buffers are **raw binary data** storage, useful for handling streams and file I/O.

Example:

```
const buf = Buffer.from('Hello');
console.log(buf.toString()); // Output: Hello
```

Buffers are used when dealing with binary data like images, files, and network packets.

Q6. Write short notes on:

(a) Express.js Scaffolding

Express.js **scaffolding** helps set up a structured project quickly using express-generator. It creates predefined folders for routes, views, and public assets, making development easier.

Example:

```
npx express-generator myApp
cd myApp && npm install
npm start
```

This generates a boilerplate Express.js application.

(b) Sharding in MongoDB

Sharding is a technique to distribute large datasets across multiple servers to ensure high availability and scalability. A **shard key** determines how data is split among shards.

Example:

```
sh.enableSharding("myDatabase");
sh.shardCollection("myDatabase.users", { "userId": "hashed" });
```

This enables sharding and distributes the users collection based on userId.

(c) Services Offered by MongoDB

1. **MongoDB Atlas** – Fully managed cloud database service.
2. **MongoDB Compass** – GUI tool for database management.
3. **MongoDB Realm** – Mobile database with offline sync.
4. **Sharding & Replication** – High availability & horizontal scaling.
5. **Full-Text Search** – Advanced search functionality.

(d) Add and Update Data in MongoDB

Add Data:

```
db.users.insertOne({ name: "Alice", age: 25 });
```

Update Data:

```
db.users.updateOne({ name: "Alice" }, { $set: { age: 26 } });
```

- insertOne() adds a new document.
- updateOne() modifies an existing document.