INP-2024-May-PYQ Answers

Q1. [10 Marks]

a. Explain HTTP and HTTPS protocol.

b. You have been tasked to develop software for Hotel management. How will you divide functionality between the front end, back end, and database?

Q2. [10 Marks]

a. Explain different types of components in React JS with an example.

b. Write a program in ReactJS to print “Hello World” on the browser.

Q3. [10 Marks]

a. Explain the Event loop in JavaScript.

Q4. [10 Marks]

a. Write a Node JS program for following: (10 marks)

   i. Create a new file and add data into it.

   ii. Append more data in the same file at the end of existing data.

   iii. Read the file data without getting the buffer data.

   iv. Rename the file.

   v. Delete the file.

b. Explain event handling in Nodejs.

Q5. [10 Marks]

a. Explain routing in Express JS along with an example.

b. Give a difference between XML and JSON.

Q6. [20 Marks]

a. Write a short note on any 4

- REPL
- React Props
- Cookies
- Session

Q1. [10 Marks] - Answers

a. Explain HTTP and HTTPS protocol.

What is HTTP?

- **HTTP (HyperText Transfer Protocol)** is the foundation of data communication on the web.
- It is a **stateless**, **application-level protocol** used to transfer data between a client (browser) and a server.
- HTTP uses **port 80** by default and does **not encrypt** the data being transmitted.

How HTTP Works:

1. The user enters a URL in the browser.
2. The browser sends an **HTTP request** to the server.
3. The server processes the request and sends back an **HTTP response** (HTML, CSS, JS, etc.).
4. The browser renders the response for the user.

What is HTTPS?

- **HTTPS (HyperText Transfer Protocol Secure)** is the **secure version of HTTP**.

- It uses **SSL/TLS encryption** to protect data during transmission.
- HTTPS uses **port 443** by default and ensures **confidentiality**, **integrity**, and **authentication**.

How HTTPS Works:

1. Before sending data, the browser and server perform a **TLS handshake**.
2. A **digital certificate** verifies the server's identity.
3. Data is encrypted before transmission, making it unreadable to third parties.

b. You have been tasked to develop software for Hotel management. How will you divide functionality between the front end, back end, and database?

Front End (Client-Side)

The front end handles the **user interface** and **user experience**. It should be intuitive, responsive, and role-based (e.g., admin, receptionist, guest).

Key Responsibilities:

- **Dashboard UI** for staff and management
- **Room booking interface** for guests
- **Login and registration forms**
- **Calendar view** for reservations
- **Payment gateway integration**
- **Real-time notifications** (e.g., booking confirmation)

Technologies:

- HTML, CSS, JavaScript

- React.js or Angular for dynamic UI
- Bootstrap or Tailwind for styling

Back End (Server-Side)

The back end handles **business logic**, **data processing**, and **communication with the database**.

Key Responsibilities:

- **Authentication and authorization**
- **Room availability logic**
- **Booking and cancellation APIs**
- **Invoice generation**
- **Staff management**
- **Email/SMS notifications**
- **Integration with third-party services** (e.g., payment, maps)

Technologies:

- Node.js with Express.js
- RESTful API design
- JWT for secure authentication

Database (Data Storage)

The database stores all **persistent data** securely and efficiently.

Key Responsibilities:

- **Guest details** (name, contact, ID proof)

- **Room inventory** (room type, status, pricing)
- **Booking records** (check-in/out dates, payment status)
- **Staff information**
- **Transaction history**
- **Feedback and reviews**

Technologies:

- **Relational DB**: MySQL or PostgreSQL for structured data
- **NoSQL DB**: MongoDB for flexible document storage
- Use of **ORMs** like Sequelize or Mongoose for data modeling

Q2. [10 Marks] - Answers

a. Explain different types of components in React JS with an example.

What Are Components in React?

- Components are the **building blocks** of a React application.
- They allow developers to split the UI into **reusable**, **independent**, and **manageable** pieces.
- React supports two main types of components: **Functional** and **Class-based**.

1. **Functional Components**

- These are **JavaScript functions** that return JSX.
- They are **stateless** by default but can use **hooks** (like `useState`, `useEffect`) to manage state and side effects.

Example:

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}
```

- With hooks:

```
import { useState } from 'react';

function Counter() {
  const [count, setCount] = useState(0);
  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={() => setCount(count + 1)}>Increment</button>
    </div>
  );
}
```

2. **Class Components**

- These use **ES6 classes** and extend `React.Component`.
- They support **state**, **lifecycle methods**, and **more control** over component behavior.

Example:

```
import React, { Component } from 'react';

class Welcome extends Component {
  render() {
    return <h1>Hello, {this.props.name}</h1>;
```

```
  }
}
```

- With state and lifecycle:

```
class Counter extends Component {
  constructor() {
    super();
    this.state = { count: 0 };
  }

  increment = () => {
    this.setState({ count: this.state.count + 1 });
  };

  render() {
    return (
      <div>
        <p>Count: {this.state.count}</p>
        <button onClick={this.increment}>Increment</button>
      </div>
    );
  }
}
```

b. Write a program in ReactJS to print “Hello World” on the browser

Steps to Create a React App

1. **Install Node.js**

Make sure Node.js is installed. You can check by running:

```
node -v
npm -v
```

If not installed, download it from https://nodejs.org

2. **Create a New Vite Project**

```
npm create vite@latest hello-world-vite
```

- When prompted:
  - **Project name:** `hello-world-vite`
  - **Framework:** `React`
  - **Variant:** `JavaScript` (or `TypeScript` if you prefer)

3. **Navigate to the Project Folder**

```
cd hello-world-vite
```

4. **Install Dependencies**

```
npm install
```

5. **Start the Development Server**

```
npm run dev
```

- Open your browser and go to `http://localhost:5173`

6. **Edit the App Component**

Open `src/App.jsx` and replace the content with:

```
function App() {
  return (
    <div>
      <h1>Hello World</h1>
    </div>
  );
}

export default App;
```

Q3. [10 Marks] - Answers

a. Explain the Event loop in JavaScript.

What is the Event Loop?

- The **Event Loop** is a core mechanism in JavaScript that handles **asynchronous operations** like callbacks, promises, timers, and I/O.
- JavaScript is **single-threaded**, meaning it executes one task at a time. The Event Loop enables it to handle **non-blocking tasks** efficiently.

How It Works (Step-by-Step)

1. **Call Stack**
    - JavaScript executes code in a stack-like structure called the **call stack**.
    - Functions are pushed onto the stack and popped off when completed.

2. **Web APIs / Background Tasks**

- Asynchronous tasks (e.g., `setTimeout`, `fetch`) are handled by the browser or Node.js environment.
- These tasks run outside the call stack and wait until they're ready.

3. **Callback Queue / Task Queue**
   - Once an async task completes, its callback is placed in the **callback queue**.

4. **Event Loop**
   - The Event Loop constantly checks:
     - If the call stack is empty
     - If there are tasks in the callback queue
   - If both conditions are met, it **moves the callback** from the queue to the call stack for execution.

Example:

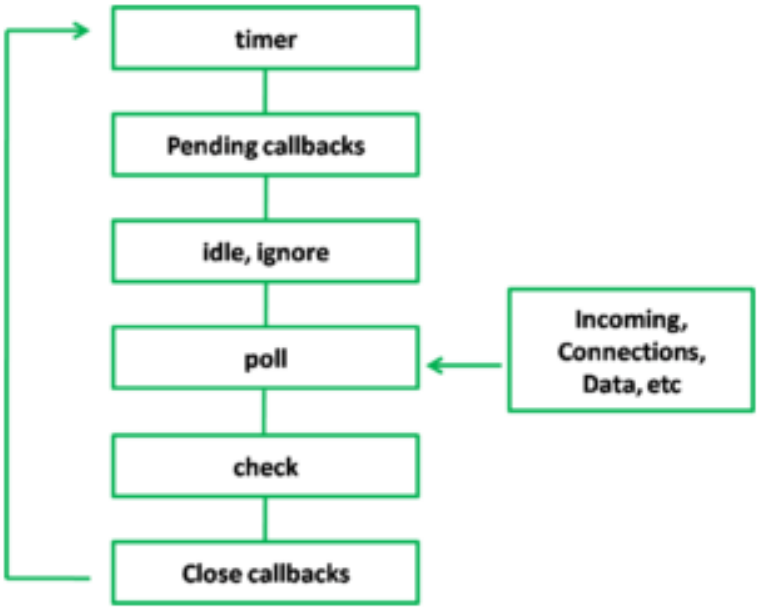
```
console.log("Start");

setTimeout(() => {
  console.log("Delayed");
}, 1000);

console.log("End");
```

- `setTimeout` is handled asynchronously.
- "Delayed" is printed after the synchronous code finishes.

Phases of Event Loop diagram:



Q4. [10 Marks] - Answers

a. Write a Node JS program for following: (10 marks)

i. Create a new file and add data into it.

ii. Append more data in the same file at the end of existing data.

iii. Read the file data without getting the buffer data.

iv. Rename the file.

v. Delete the file.

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

// 1️⃣ Create a new file and add data
fs.writeFile('sample.txt', 'This is the first line.\n', (err) => {
  if (err) throw err;
  console.log('File created and data added.');
```

```
// 2️⃣ Append more data
fs.appendFile('sample.txt', 'This is the appended line.\n', (err) => {
  if (err) throw err;
  console.log('Data appended successfully.');

  // 3️⃣ Read file data (without buffer)
  fs.readFile('sample.txt', 'utf8', (err, data) => {
    if (err) throw err;
    console.log('File content:\n', data);

    // 4️⃣ Rename the file
    fs.rename('sample.txt', 'renamedSample.txt', (err) => {
      if (err) throw err;
      console.log('File renamed successfully.');

      // 5️⃣ Delete the file
      fs.unlink('renamedSample.txt', (err) => {
        if (err) throw err;
        console.log('File deleted successfully.');
      });
    });
  });
});
```

b. Explain event handling in Nodejs.

What is Event Handling in Node.js?

- Node.js uses an **event-driven architecture**, which means it reacts to events (like data received, file opened, or user input) using **callbacks**.

- It is built on the **Event Loop** and **EventEmitter** class, allowing asynchronous and non-blocking operations.

How Event Handling Works

1. **EventEmitter Class**
   - Provided by the `events` module in Node.js..
   - Allows you to create, emit, and listen to custom events.

2. **Registering an Event Listener**
   - Use `.on()` or `.addListener()` to listen for an event.

3. **Emitting an Event**
   - Use `.emit()` to trigger an event and execute its associated callback.

Example: Basic Event Handling

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

// Registering an event listener
myEmitter.on('greet', () => {
  console.log('Hello, Sameer!');
});

// Emitting the event
myEmitter.emit('greet');
```

Q5. [10 Marks] - Answers

a. Explain routing in Express JS along with an example.

What is Routing in Express.js?

- **Routing** refers to defining how an application responds to **client requests** for specific URLs or paths.
- In Express.js, routing is handled using methods like `.get()`, `.post()`, `.put()`, and `.delete()` which correspond to HTTP methods.
- Each route can have a **path**, a **callback function**, and optional **middleware**.

Syntax of a Route

```
app.METHOD(PATH, HANDLER)
```

- `METHOD`: HTTP method (GET, POST, etc.)
- `PATH`: URL path (e.g., `/home`)
- `HANDLER`: Function that executes when the route is matched

Example: Basic Routing

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

// GET route
app.get('/', (req, res) => {
  res.send('Welcome to the Hotel Management System');
});

// POST route
app.post('/book-room', (req, res) => {
  res.send('Room booked successfully');
});
```

```
// PUT route
app.put('/update-room/:id', (req, res) => {
  res.send(`Room ${req.params.id} updated`);
});

// DELETE route
app.delete('/cancel-booking/:id', (req, res) => {
  res.send(`Booking ${req.params.id} cancelled`);
});

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

b. Give a difference between XML and JSON.

Feature	XML (eXtensible Markup Language)	JSON (JavaScript Object Notation)
Syntax Style	Tag-based (uses opening and closing tags)	Key-value pairs with braces and brackets
Readability	More verbose and harder to read	Lightweight and easier to read
Data Format	Document-oriented	Data-oriented
Parsing Speed	Slower due to complex structure	Faster and more efficient
Support for Data Types	Limited (everything is text)	Supports strings, numbers, arrays, booleans
Schema Support	Supports DTD and XSD for validation	Uses JSON Schema (less strict)
Use Case	Used in legacy systems, configuration	Widely used in APIs and web applications

Feature	XML (eXtensible Markup Language)	JSON (JavaScript Object Notation)
Comments Support	Supports comments	Does not support comments

Q6. [20 Marks] - Answers

Write a short note on any 4:

- REPL
- React Props
- Cookies
- Session

1. REPL (Read–Eval–Print–Loop)

Definition

REPL stands for **Read–Eval–Print–Loop**. It is an **interactive programming environment** that takes user input, executes it, and returns the output immediately.

It is commonly used in environments like **Node.js**, **Python shell**, and **browser consoles** for testing and debugging code quickly.

Working Process

1. **Read** – Takes the user's input (JavaScript code).
2. **Eval** – Evaluates the entered code.
3. **Print** – Displays the result of evaluation.
4. **Loop** – Repeats the process for the next command.

Example (Node.js REPL)

```
> node
> let x = 5
> x * 2
10
```

Advantages

- Quick testing and debugging.
- No need to create or run separate files.
- Great for learning and experimenting with JavaScript.

---

2. React Props

Definition

Props (short for *properties*) are **read-only inputs** passed from a **parent component** to a **child component** in React.

They make components **dynamic and reusable**.

Key Features

- Immutable (cannot be changed by child components).
- Allow **data and function sharing** between components.
- Enable **component reusability** and modular UI design.

Example

```
function Welcome(props) {
  return <h2>Hello, {props.name}!</h2>;
}

function App() {
```

```
  return <Welcome name="Sameer" />;
}
```

Here, `name` is a **prop** passed from the parent (`App`) to the child (`Welcome`).

Advantages

- Promotes **component reusability**.
- Ensures **unidirectional data flow** (parent → child).
- Keeps components **clean and predictable**.

---

3. Cookies

Definition

A **cookie** is a small piece of **data stored on the client's browser** by a web server.

It helps websites remember **user preferences, sessions, and login information** across visits.

Key Features

- Stored as **name–value pairs**.
- Sent automatically with every HTTP request to the same domain.
- Can have an **expiration date**.
- Typically limited to **4 KB** of data per cookie.

Example (JavaScript)

```
document.cookie = "username=Sameer; expires=Fri, 31 Dec 2025 12:00:00 UTC; path=/";
```

Uses

- User authentication (login sessions).
- Tracking user activity.
- Saving site preferences.

---

4. Session

Definition

A **session** is a **temporary data storage mechanism** used by a web server to maintain user information while they interact with a website.

It typically lasts until the **browser is closed** or **session timeout** occurs.

Key Characteristics

- Stores user-specific data (e.g., user ID, cart items).
- Data is stored on the **server**, not the client.
- Each user is assigned a **unique session ID**.
- More **secure** than cookies since data isn't exposed to the client.

Example (Express.js)

```
req.session.username = "Sameer";  // Setting session data
console.log(req.session.username); // Accessing session data
```

Uses

- User authentication and authorization.
- Shopping cart management.
- Preserving state between multiple page requests.