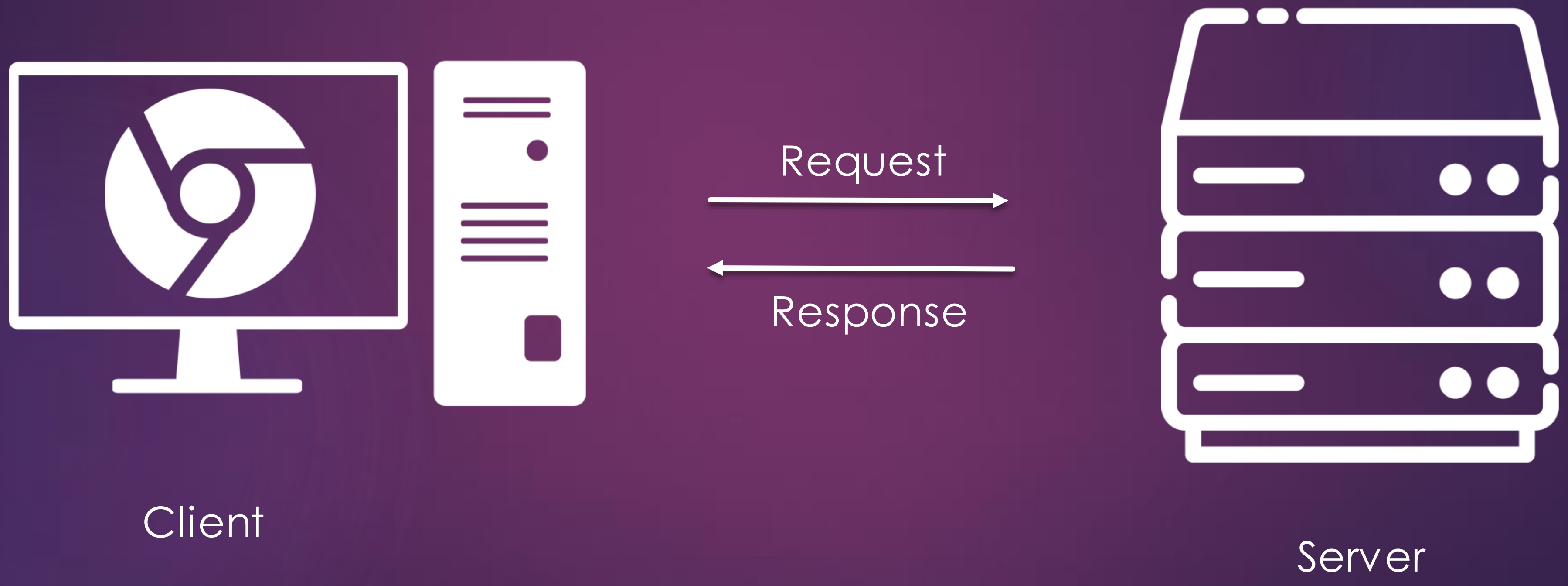
Express JS

BY CODE EATER

Website



Server

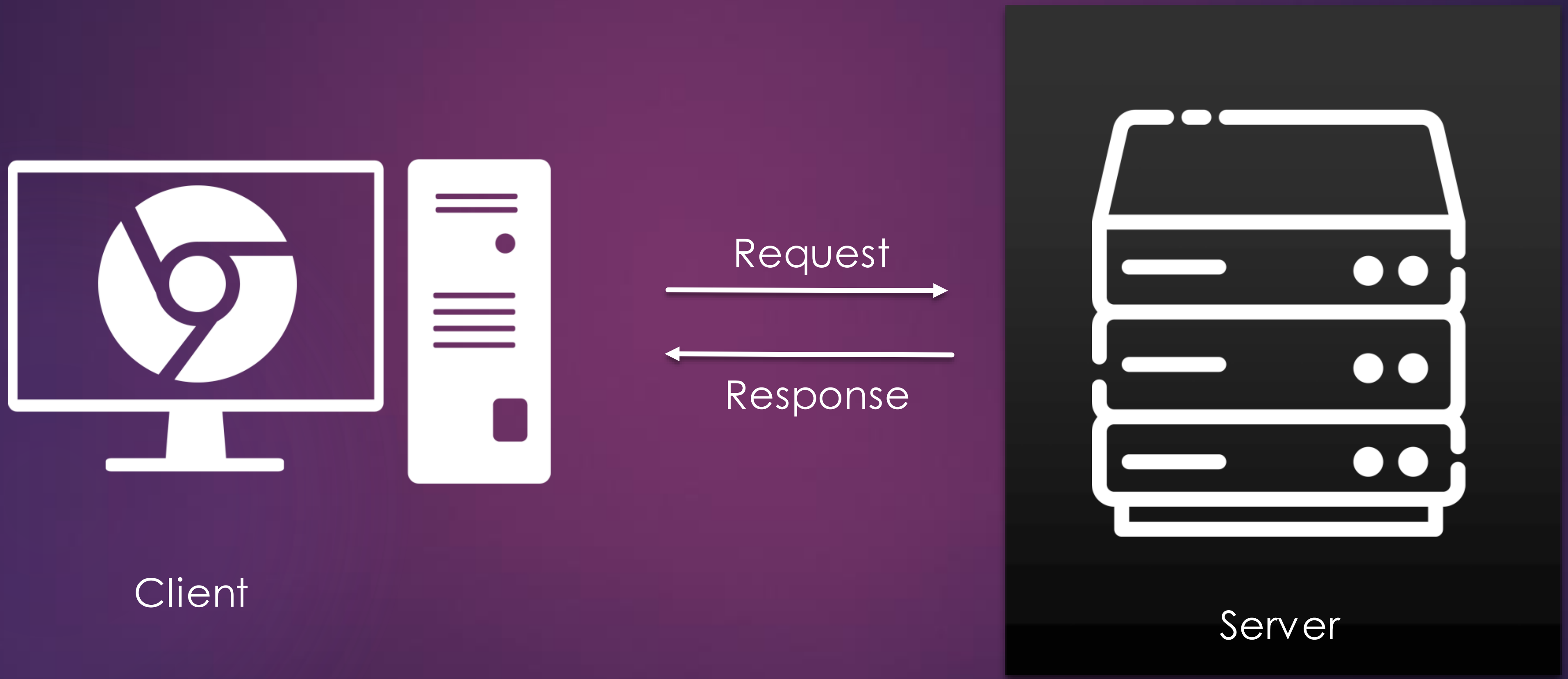
- A server is a computer program or hardware device that provides services, resources, or data to other computers, devices, or clients over a network.
- Servers play a crucial role in networked computing environments, enabling communication, data sharing, and various functionalities that clients (such as user devices or other servers) may require.

Types of Server

- **Web Server:** Web servers deliver web content (such as HTML, images, and files) to clients, typically web browsers. They handle HTTP requests and responses, serving web pages and files to users
- **Application Server:** Application servers host and manage software applications, providing services like authentication, data access, and business logic execution
- **Database Server:** Database servers store and manage databases, handling requests to read or update data.

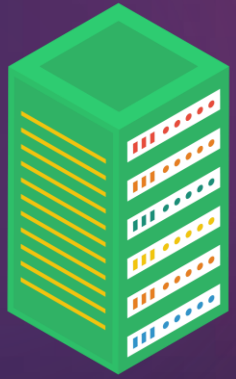
- **File Server:** File servers store and manage files that clients can access and share over a network
- **Mail Server:** Mail servers handle email communication, sending, receiving, and storing emails for users.

Website



Node JS

- Node.js is an open-source, server-side runtime environment that allows you to execute JavaScript code on the server.
- Node.js enables developers to build scalable and high-performance network applications using JavaScript, which was traditionally known for its use in web browsers.


Server

JavaScript

Browser

Express JS

- Express.js, commonly referred to as Express, is a minimalistic and flexible web application framework for Node.js.

- Express.js is designed to simplify the process of creating server-side applications by providing a structured framework for handling routes, middleware, and various HTTP requests and responses.

Sever in Express JS vs NodeJS

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

const server = http.createServer((req, res) => {
  res.writeHead(200, { 'Content-Type': 'text/plain' });
  res.end('Hello, Node.js HTTP Server!\n');
});

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

Node JS

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

const app = express();

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

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

Express JS

What is PORT?

- A "port" refers to a communication endpoint or a logical construct used to identify a specific process or service running on a computer. Ports are an essential concept for enabling communication between different computers over a network, such as the internet.

- **Port Numbers:** Ports are identified by numbers, called "port numbers," which range from 0 to 65,535. These numbers are used to differentiate between various network services or processes on a single computer.

Get Vs Post

- GET and POST are two of the most common HTTP methods used to send and receive data between a client (usually a web browser) and a server.
- The GET method is primarily used to request data from a server. It retrieves information specified in the URL's query parameters and does not modify any data on the server.
- The POST method is used to submit data to be processed to a specified resource. It can modify data on the server or trigger server-side actions.

req and res

In Express.js, **req** and **res** are objects that represent the HTTP request and response, respectively, for a particular incoming HTTP request made to your server. These objects are provided by Express.js and are passed as arguments to route handler functions when a request matches a specific route.

Req object

The **req** object contains information about the incoming HTTP request, including details about the client, the requested URL, headers, query parameters, form data, and more. It provides a way for your route handler to access and process the data sent by the client.

Common properties and methods of the **req** object include:

- **req.params**: An object containing parameters from the route URL, useful for capturing dynamic segments in the URL.
- **req.query**: An object containing query parameters from the URL.
- **req.body**: The parsed request body (when body parsing middleware is used), usually used for handling form data or JSON payloads.
- **req.headers**: An object containing the headers sent by the client.
- **req.cookies**: An object containing cookies sent by the client.
- **req.method**: The HTTP method (GET, POST, PUT, etc.) of the request.
- **req.path**: The path portion of the URL.
- **req.url**: The full URL of the request.
- **req.params**: An object containing route parameters from the URL.

Res object

The **res** object is used to construct and send the HTTP response back to the client. It provides methods and properties for setting headers, status codes, sending data, and more.

Common properties and methods of the **req** object include:

- **res.send(data)**: Sends the specified data as the response body.
- **res.json(data)**: Sends JSON-formatted data as the response body.
- **res.status(code)**: Sets the HTTP status code of the response.
- **res.setHeader(header, value)**: Sets an HTTP header for the response.
- **res.cookie(name, value, options)**: Sets a cookie in the response.
- **res.redirect(url)**: Redirects the client to the specified URL.
- **res.render(view, data)**: Renders an HTML view using a template engine (like EJS or Pug).
- **res.locals**: An object that holds data scoped to the current request-response cycle, often used to pass data to view templates.

Middle Ware

- Middleware refers to a software component that sits between two other software components or systems, acting as a bridge or intermediary.
- In the context of web development and frameworks like Express.js, middleware functions are used to process and manipulate HTTP requests and responses before they reach their final destination (route handler or client) or after they leave the final destination.

Basic Structure

```
app.get([path], middlewareFunction)
```

```
app.post([path], middlewareFunction)
```

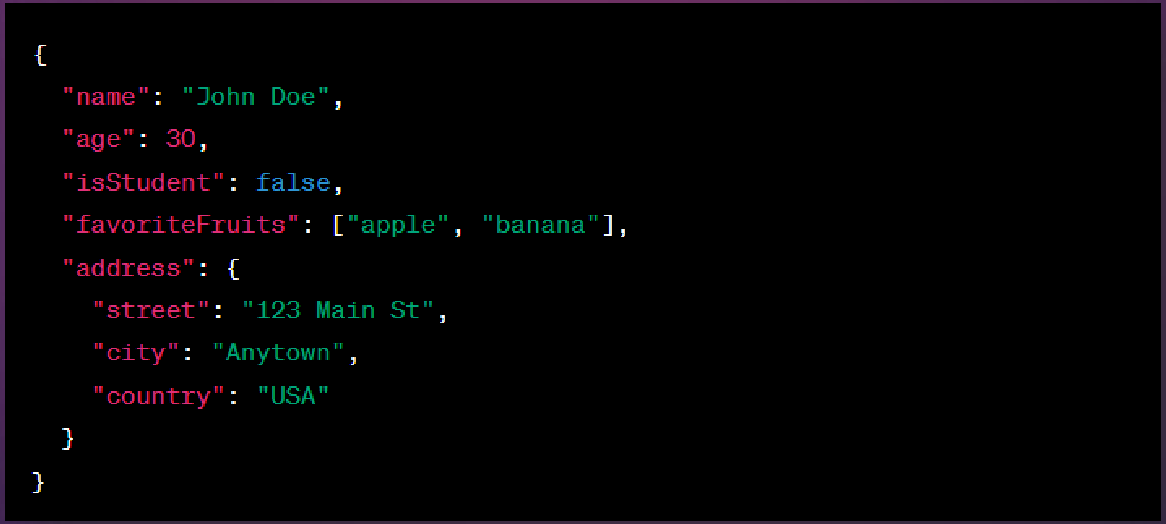
```
app.use([path], middlewareFunction)
```

Nodemon

- **nodemon** is a popular utility in the Node.js ecosystem that helps developers in automatically restarting the Node.js application when changes are detected in the source code.

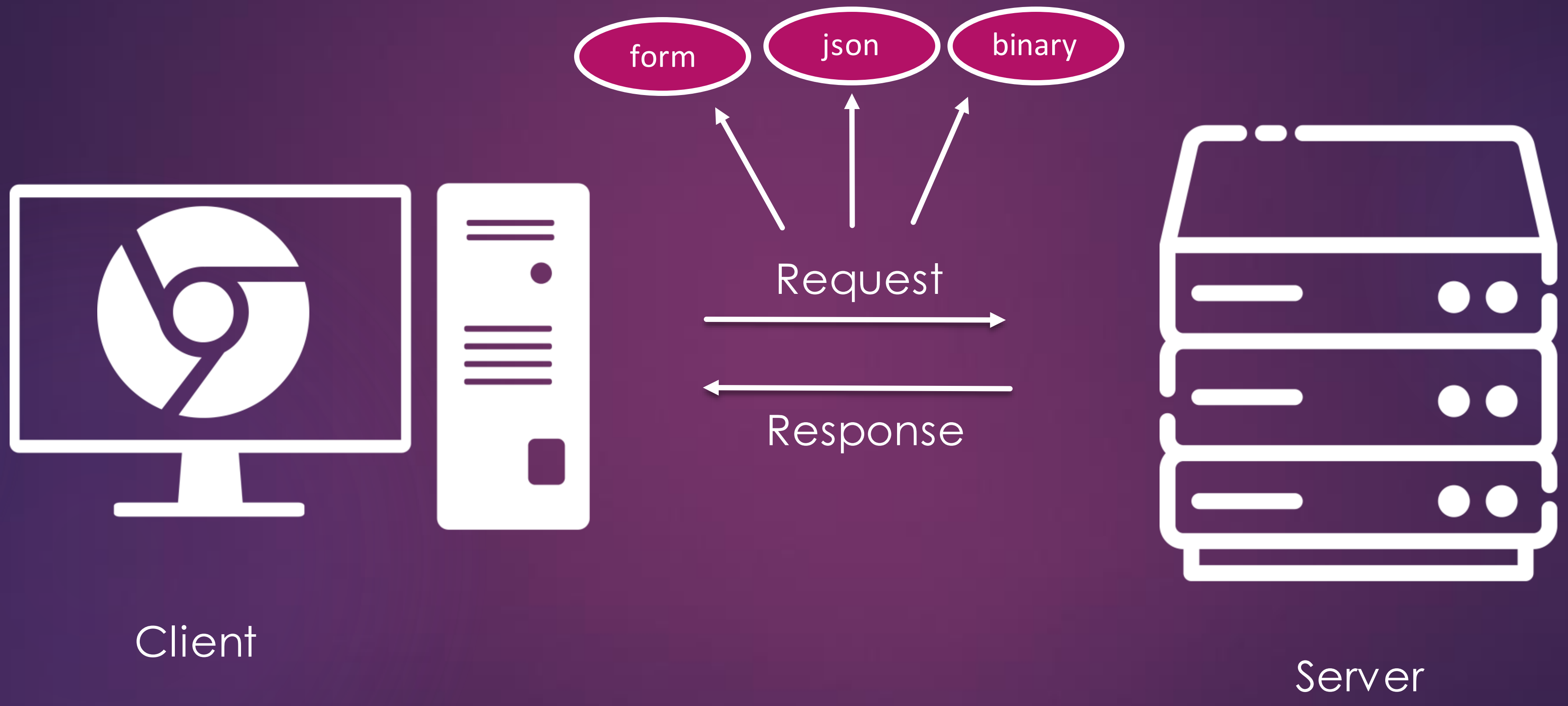
Handling Routes

- Order of request matters.
- / is not if



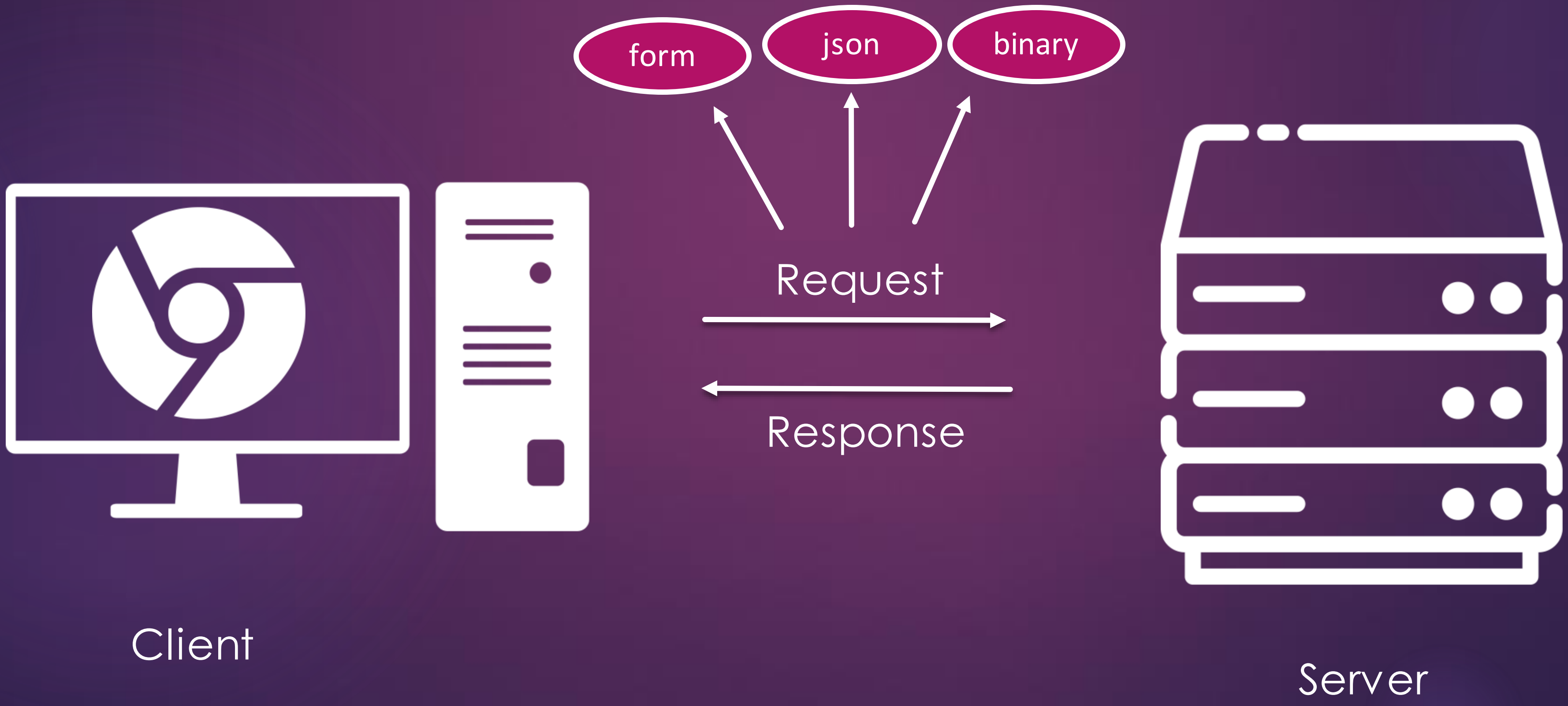
```
{
  "name": "John Doe",
  "age": 30,
  "isStudent": false,
  "favoriteFruits": ["apple", "banana"],
  "address": {
    "street": "123 Main St",
    "city": "Anytown",
    "country": "USA"
  }
}
```

Ways of parsing request



Parsing

Parsing refers to the process of analyzing or interpreting structured data in order to extract meaningful information from it.



Middlewares for Parsing

form	bodyParser.urlencoded()
json	Express.json()
raw	express.raw()
text	express.text()

bodyParser.urlencoded()

- **bodyParser.urlencoded()** is a middleware provided by the **body-parser** library, which is commonly used in Express.js applications to parse data from the body of incoming HTTP requests.

- This middleware specifically parses data submitted via HTML forms with the **application/x-www-form-urlencoded** content type.

Headers

- In Express.js, HTTP headers are pieces of metadata that provide additional information about an HTTP request or response.
- Headers are used to convey important details, such as the content type, caching information, authentication tokens, and more.
- Headers are sent along with the HTTP request from the client to the server and with the HTTP response from the server to the client.

Headers

```
app.get('/content', (req, res) => {
  res.setHeader('Content-Type', 'text/plain');
  res.send('This is plain text content.');
});
```

Response in text

```
app.get('/json', (req, res) => {
  const data = {
    message: 'Hello, JSON!'
  };

  // Set the Content-Type header to indicate JSON data
  res.setHeader('Content-Type', 'application/json');

  // Send the JSON data as a response
  res.send(JSON.stringify(data));
});
```

Response in JSON

Headers

Common HTTP headers:

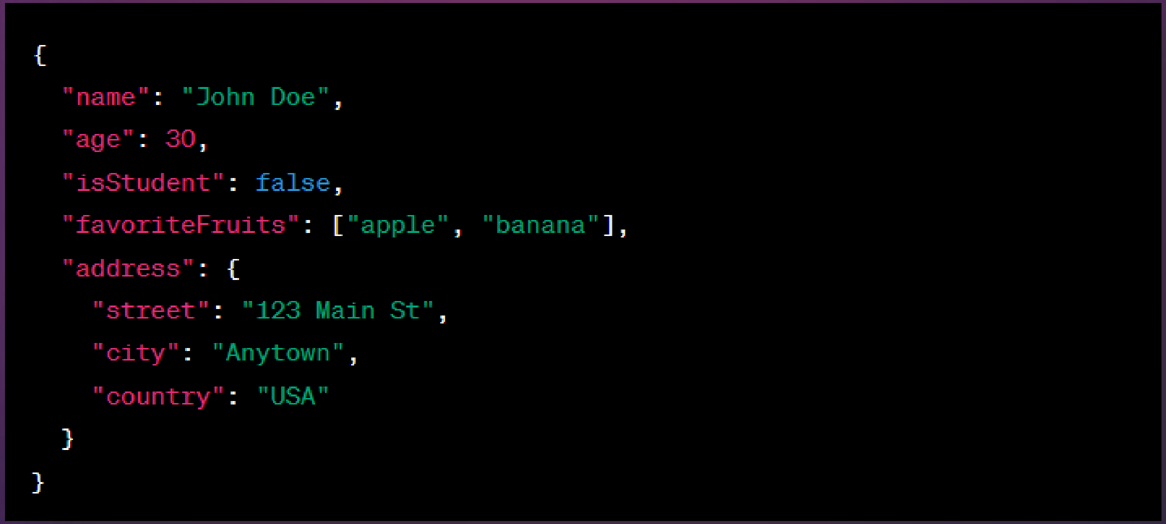
- **Content-Type:** Specifies the format of the content being sent or received (e.g., JSON, HTML, plain text).
- **Authorization:** Used to send authentication tokens or credentials for access control.
- **User-Agent:** Indicates the client application or browser making the request.
- **Cache-Control:** Controls caching behavior to optimize network performance.
- **Location:** Used for redirection, indicating the URL to which the client should be redirected.
- **Set-Cookie:** Used for setting cookies in the response.
- **Access-Control-Allow-Origin:** Specifies which origins are allowed to access the resource in a cross-origin request.

Body

- The "body" refers to the data that is sent in the request's body from the client to the server or from the server to the client.
- The body can contain various types of data, such as JSON, form data, plain text, or even binary data like images or files.
- When a client sends data to the server, such as when submitting a form or making an API request, the data is typically included in the body of the request. Similarly, when the server sends a response to the client, it may include data in the response body.

Json

- JSON (JavaScript Object Notation) is a lightweight data interchange format that is easy for humans to read and write and easy for machines to parse and generate.
- It is widely used for transmitting data between a server and a client, as well as for storing configuration data and structuring information in various applications.



```json
{
  "name": "John Doe",
  "age": 30,
  "isStudent": false,
  "favoriteFruits": ["apple", "banana"],
  "address": {
    "street": "123 Main St",
    "city": "Anytown",
    "country": "USA"
  }
}
```

Thank You

Website – www.codeeater.in

Instagram - @codeeater21

LinkedIn – @kshitijweb3

Twitter - @kshitijweb3