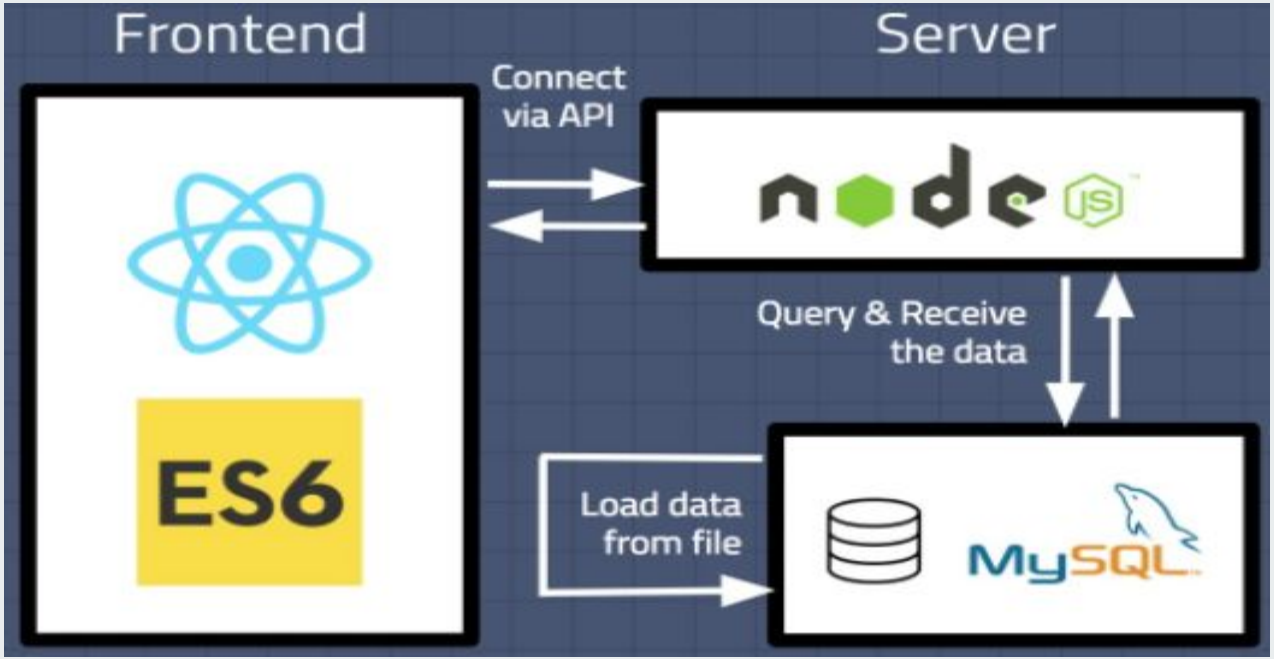
NodeJS

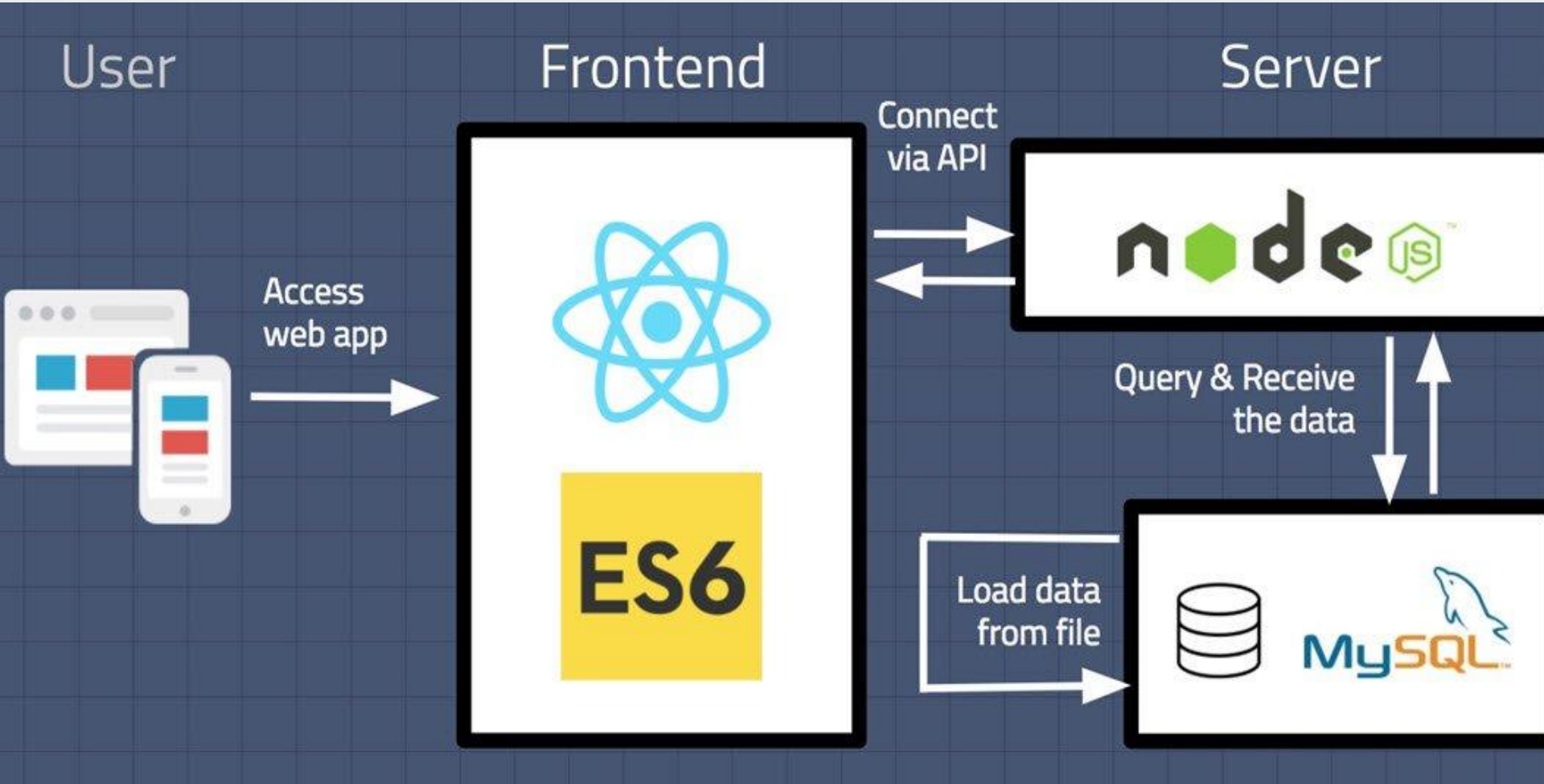
Vivek Sharma
vksh397@gmail.com

Client-Server Architecture:



Vivek Sharma
vksh397@gmail.com

Client-Server Architecture



NodeJS:

What is NodeJS?

- NodeJS is an open source server environment
- NodeJS is free to use
- Node JS runs on various platforms (Windows, Linux, Unix, Mac OS X, etc.)
- NodeJS uses JavaScript on the server
- Easy to learn for JS developer as using JS on the server side.

Benefit of NodeJS

- Node.js uses asynchronous programming!

Example: Compare with other server side programmings:

A common task for a web server can be to open a file on the server and return the content to the client.

Here is how PHP or ASP handles a file request:

1. Sends the task to the computer's file system.
2. Waits while the file system opens and reads the file.
3. Returns the content to the client.
4. Ready to handle the next request.

Here is how Node.js handles a file request:

1. Sends the task to the computer's file system.
2. Ready to handle the next request.
3. When the file system has opened and read the file, the server returns the content to the client.

Benefit of NodeJS

- Node.js uses asynchronous programming!

Node.js eliminates the waiting, and simply continues with the next request.

Node.js runs single-threaded, non-blocking, asynchronous programming, which is very memory efficient.

What Can Node.js Do

Node.js I/O

- Node.js can generate dynamic page content
- Node.js can create, open, read, write, delete, and close files on the server
- Node.js can collect form data
- Node.js can add, delete, modify data in your database

Node.js File

- Node.js files contain tasks that will be executed on certain events.
- A typical event is someone trying to access a port on the server.
- Node.js files must be initiated on the server before having any effect
- Node.js files have extension ".js"

Data Type

JavaScript has 8 Data Types:

1. String
2. Number
3. Bigint
4. Boolean
5. Undefined
6. Null
7. Symbol
8. Object

The Object Data Type:

The object data type can contain:

1. An object
2. An array
3. A date

Variables

Variables are Containers for Storing Data and use them.

JavaScript Variables can be declared in 4 ways:

Automatically

*Using **var***

*Using **let***

*Using **const***

To declare variables.

a = 5; //automatically

var b= 5;

let c = 6;

const d = 7;

NodeJS Variables

In a programming language, variables are used to store data values.

Example:

var a = 5;

An equal sign is used to assign values to variables.

In this example, a is defined as a variable. Then, a is assigned (given) the value 5.

- The ***var*** keyword was used in all JavaScript code from 1995 to 2015.
- The ***let*** and const keywords were added to JavaScript in 2015.
- The ***var*** keyword should only be used in code written for older browsers.

Variables

*When to Use **var, let, or const**?*

1. Always declare variables.

*2. Always use **const** if the value should not be changed.*

*3. Always use **const** if the type should not be changed (Arrays and Objects).*

*4. Only use **let** if you can't use const.*

*5. Only use **var** if you MUST support old browsers.*

Closures

Closures

*A closure is the combination of a function bundled together (enclosed) with references to its surrounding state (**the lexical environment**).*

In other words, a closure gives you access to an outer function's scope from an inner function. In JavaScript, closures are created every time a function is created, at function creation time.

Lexical scoping

*Consider the example code : **closure.js***

Hoisting

Hoisting:

variable declarations, wherever they occur in a script, are processed before any code within the script is executed.

Declaring a variable anywhere in the code is equivalent to declaring it at the top.

This also means that a variable can appear to be used before it's declared.

This behavior is called hoisting, as it appears that the variable declaration is moved to the top of the function, static initialization block, or script source in which it occurs.

Node Express

Hoisting:

Step 1: Install Required Dependencies

We need to install Express, a powerful and minimalist web framework, to build our API. Run the following command to install Express:

npm install express

Additionally, we will use body-parser to parse incoming request bodies and cors to handle ***Cross-Origin Resource Sharing (CORS)*** *headers:*

npm install body-parser cors

Node Express

Step 2: Create the Server

Create a new file in your project folder. This file will serve as the entry point of our API.

```
// Import required modules

const express = require('express');

const bodyParser = require('body-parser');

const cors = require('cors');

// Create an instance of Express

const app = express();

// Middleware setup

app.use(bodyParser.json());

app.use(cors());

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

Node Express API Routes

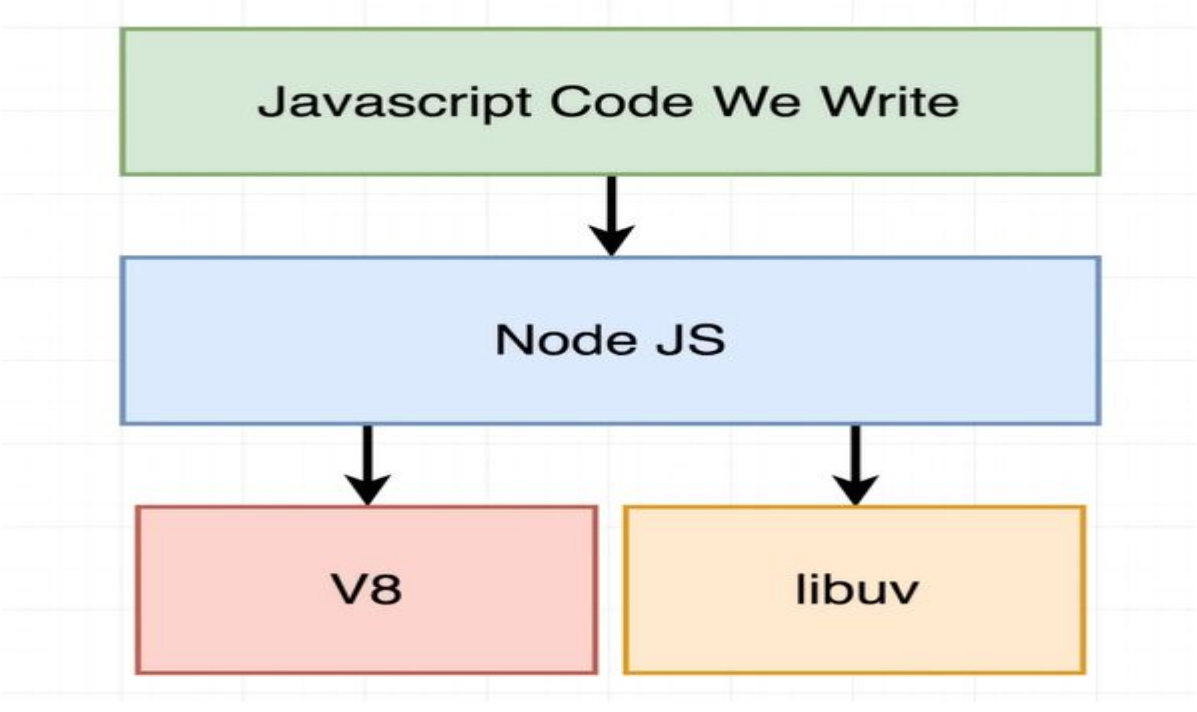
Step 4: Define API Routes

In this step, we will create routes for handling various API endpoints. For demonstration purposes, we will create routes to handle CRUD operations for a collection of users.

Create a new file named users.js inside a folder named routes in your project directory. This file will contain the API routes related to users.

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

const router = express.Router();

// Sample user data (temporary)

let users = [  { id: 1, name: 'John Doe', age: 30 },

 { id: 2, name: 'Jane Smith', age: 25 }];

// Get all users

router.get('/users', (req, res) => {

 res.json(users);

});
```

Internal working flow:



Thank You!