What is Full Stack?

Full stack refers to the complete set of technologies and tools used to build both the front-end (client-side) and back-end (server-side) parts of a web application.

A full-stack developer is someone who is proficient in both areas and capable of developing a complete web application from start to finish.

Components of Full Stack Development

Front-End (Client-Side)

Back-End (Server-Side)

Database

Version Control

DevOps and Deployment

Server Side Programming

Welcome to this journey of unraveling the captivating world of web development, where we'll delve into the intricacies of both the front-end and back-end landscapes, focusing on the empowering capabilities of Node.js.

Transitioning to Back-End Development with Node.js

Front-End vs. Back-End

Front-end development focuses on the user interface, while back-end development handles server-side logic, data storage, and communication.

Node.js: A Powerhouse

Node.js is a JavaScript runtime environment that allows for building scalable and efficient back-end applications, using the language you already know.

Front End

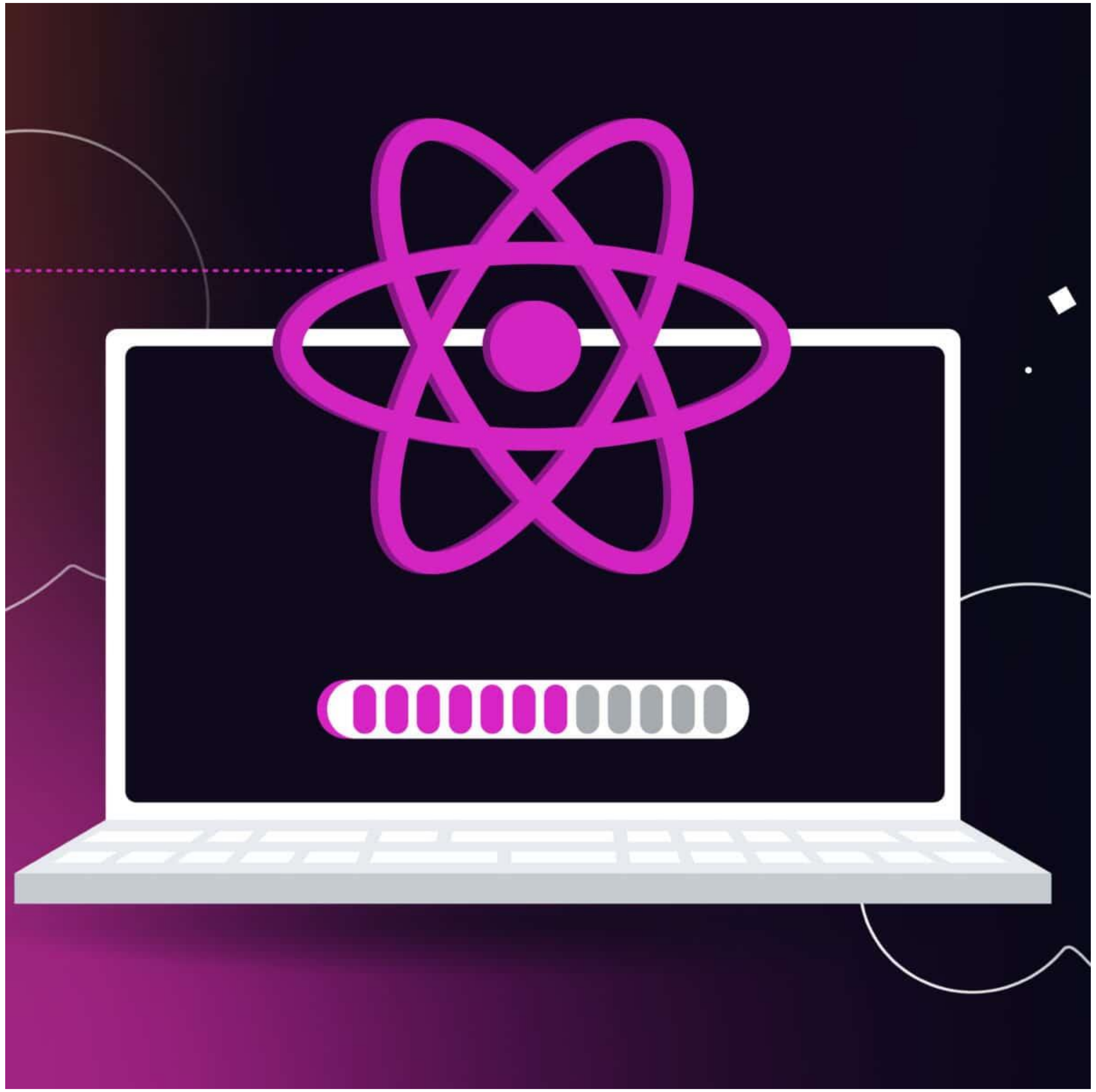
Front-end refers to the user interface and experience of a website or application (what the user interacts with).

Key Technologies:
HTML/CSS/JavaScript: Basic building blocks for front-end development.
Libraries/Frameworks: React, Angular, Vue.js.

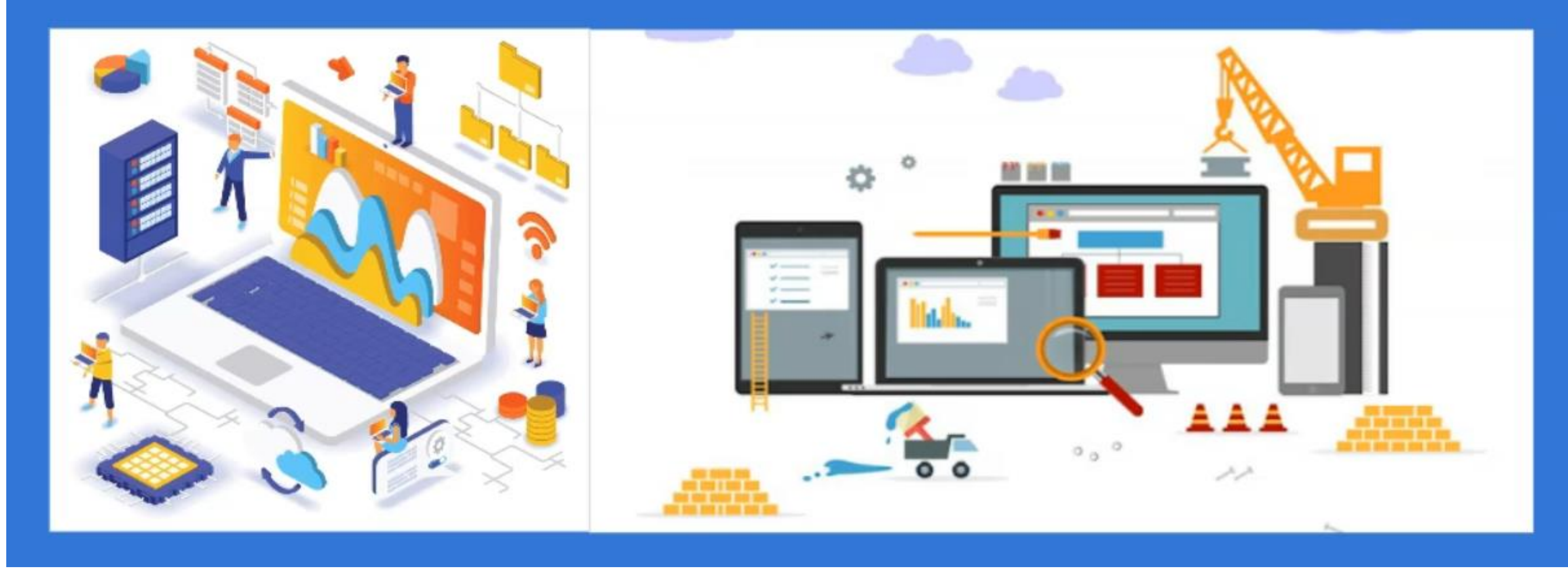
Responsibilities:

- Display content and layout.
- Handle user input and actions.
- Send requests to the back-end.

Example: A React app that fetches and displays data from an API.



Back End



Back-end refers to the server, database, and application logic that handle the data and operations behind the scenes.

Key Technologies:

Node.js, Express.js (in the MERN stack).

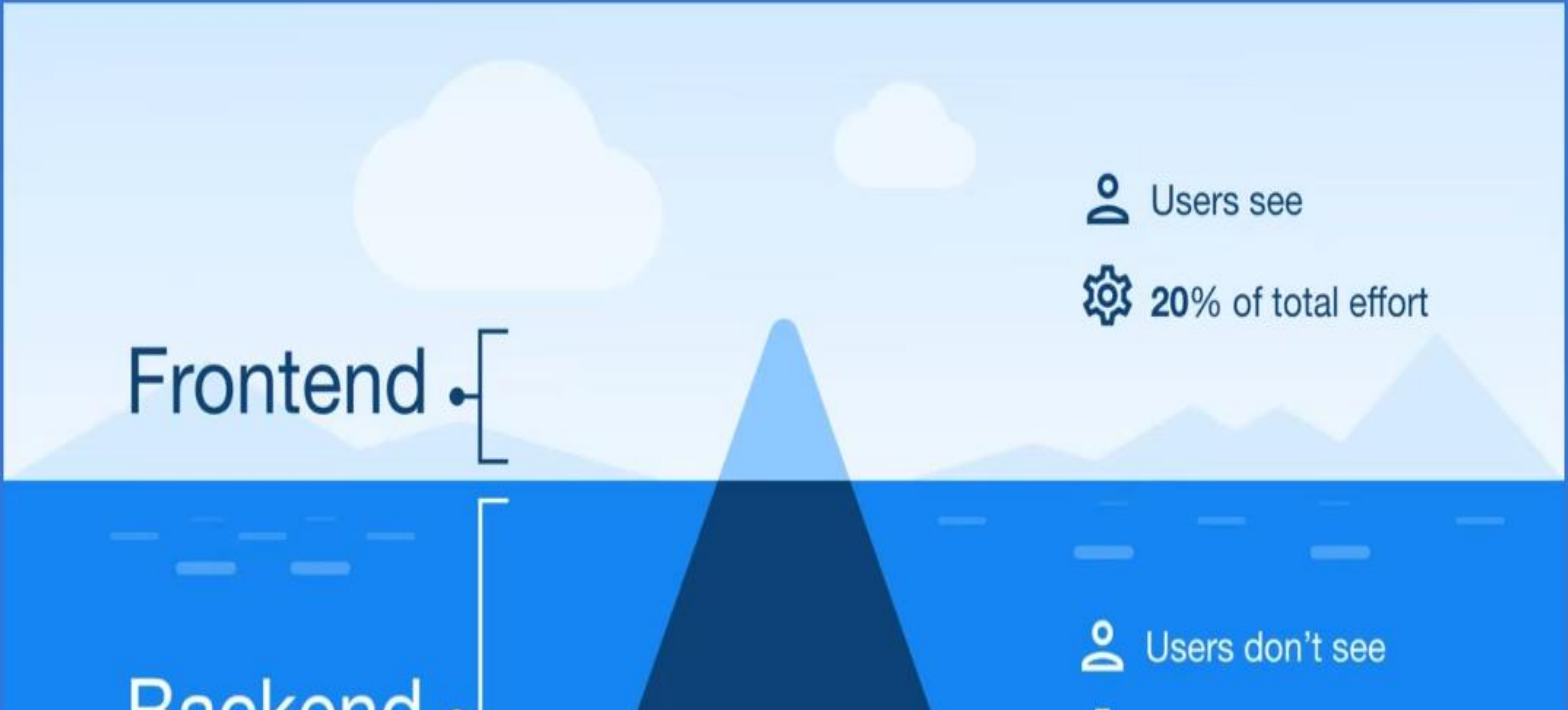
Ruby on Rails, Django, Laravel (other frameworks)

Responsibilities:

- Process user requests.
- Interact with databases.
- Provide APIs for front-end to consume.

Example: An Express.js server handling API routes and interacting with a MongoDB database.

The Back End



Gamma

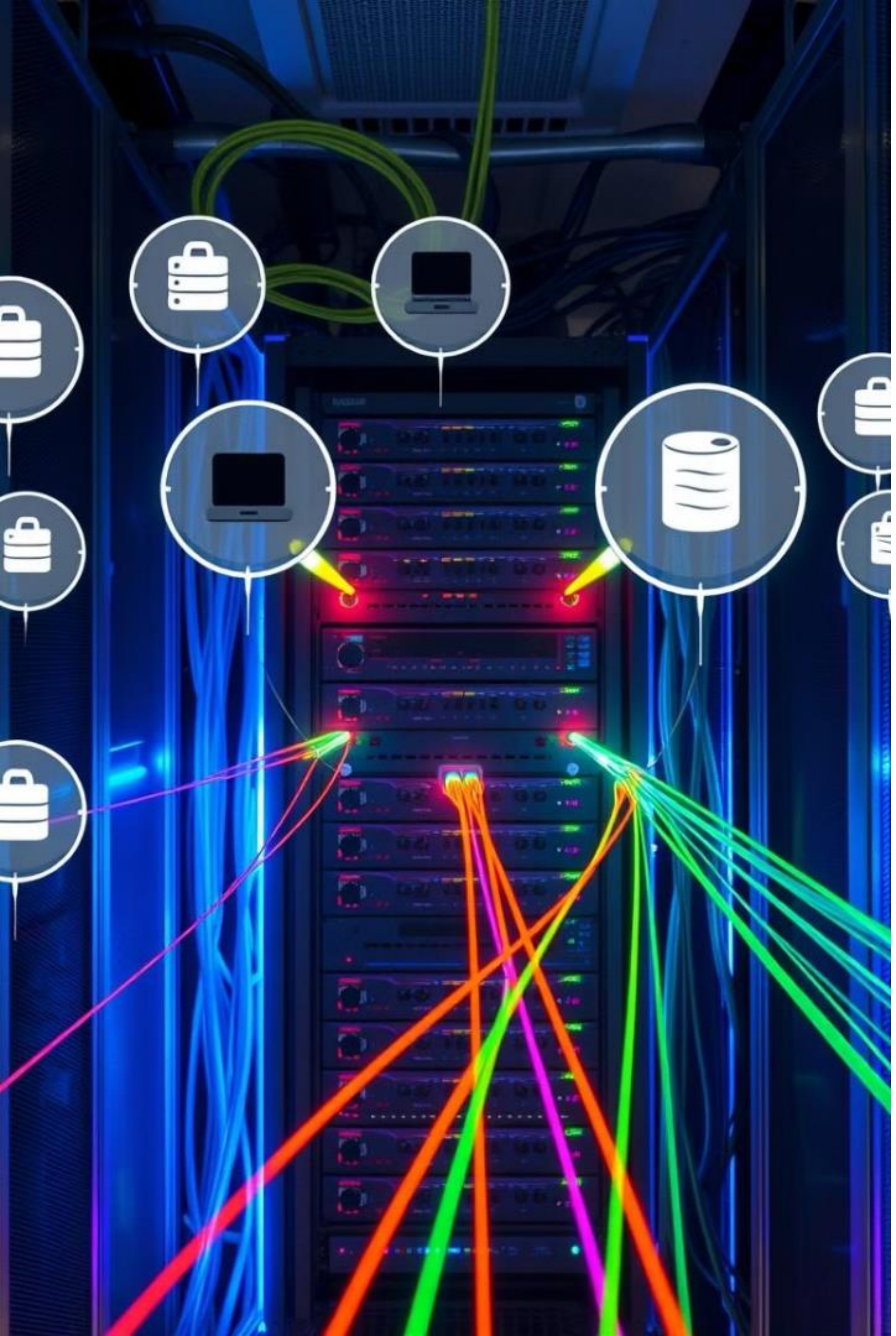
Understanding the Server in Node.js

A Server is simply a computer that listens for incoming requests

Handle requests from clients.

Process and route requests to appropriate services.

Return responses with data or content.



Exploring Application Servers and Databases

Application Server

Provides a platform for running web applications and handles communication between different components.

An application server hosts and runs backend applications, such as API servers or business logic.

Hosting and running backend services.

Managing server-side logic.

Communicating with databases or other services.

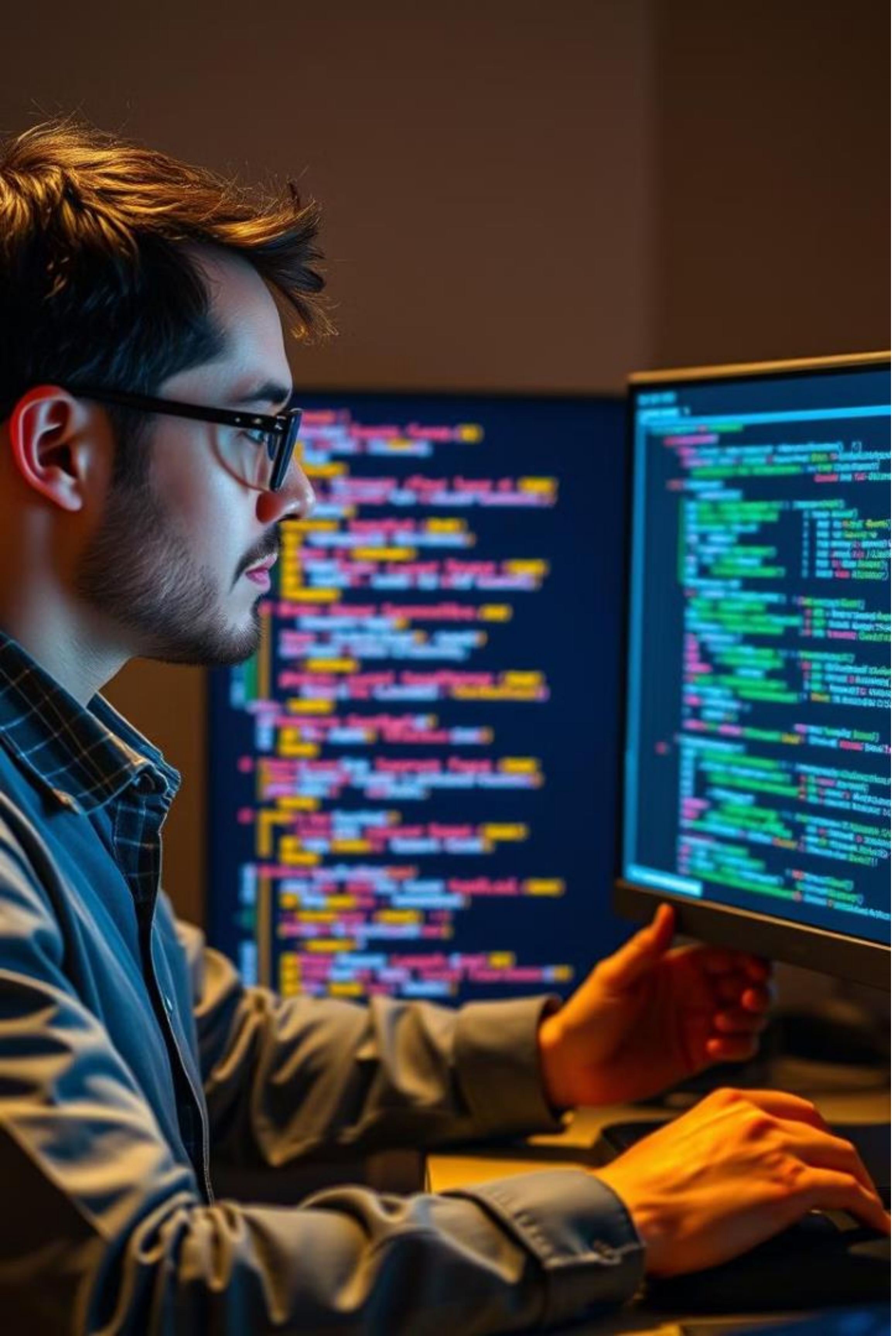
Database

Stores and manages data persistently, enabling applications to access and manipulate information efficiently.

Types of Databases

:Relational Databases: SQL-based (e.g., MySQL, PostgreSQL).

NoSQL Databases: Non-relational (e.g., MongoDB, Firebase).



Mastering API Development with Node.js

1

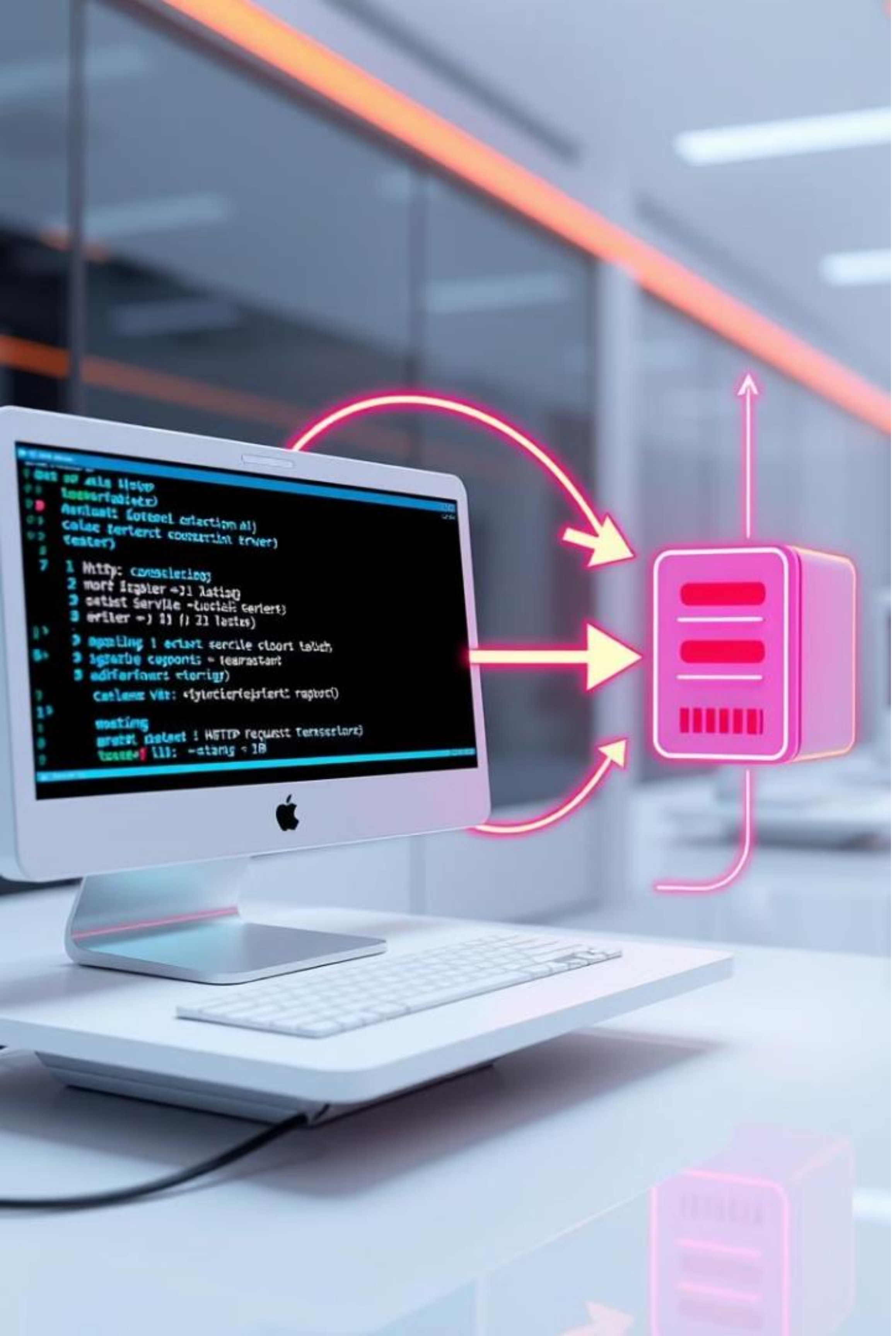
API Fundamentals

APIs define how different software components interact and communicate with each other.

2

Node.js for APIs

Node.js provides a robust framework for building RESTful APIs that are efficient, scalable, and highly performant.



Comprehending Web Communication and HTTP Requests

1 HTTP Protocol

HTTP is the foundation of web communication. It defines how web browsers request information from web servers.

2 Requests and Responses

HTTP requests are used by clients to request information from servers, while HTTP responses carry the requested data back.

Implementing HTTP Methods: GET, PUT, POST, and DELETE

GET Retrieves information from a specified resource on the server.	**PUT** Updates or replaces an existing resource on the server.
POST Creates a new resource on the server.	**DELETE** Removes a resource from the server.

Introduction to MERN Stack

MongoDB (Database)

Express.js (Back-end framework)

React (Front-end library)

Node.js (JavaScript runtime)

Why MERN?

1. Single Language Across the Stack (JavaScript)Unified Development:
2. Scalability
3. Flexibility and Performance
4. Real-Time Data and Fast Development
5. Large Community and Ecosystem
6. Full-Stack Development
7. JSON-Based Communication
8. Cost-Effective
9. Modular and Maintainable Code
10. Fast and Easy

Introduction to Node js

- A runtime environment to executing javascript pgm.
- Node.js operates on a **single thread**, but it can handle many concurrent connections thanks to its asynchronous nature.
- Node.js uses **events** to handle asynchronous operations. Instead of waiting for a task to complete (blocking), Node.js uses a callback function that gets triggered once the task is finished.
- Traditional servers wait for an operation (like reading from a file or a database query) to complete before moving on. In contrast, Node.js uses non-blocking I/O to handle operations in parallel, increasing the performance and scalability of applications.
- Node.js is cross-platform, which means it can run on various operating systems like Windows, macOS, and Linux. This makes deployment and development more flexible.
- **NPM** is the package manager for Node.js, and it is the largest ecosystem of open-source libraries.
- Node.js comes with a rich set of built-in libraries, such as HTTP, File System (FS), Events, Streams, and more, to facilitate various functionalities for developers.

Node js Installation

Download Node.js:

Visit Node.js official website.Choose the LTS (Long Term Support) version.

Install Node.js:Run the installer based on your OS (Windows, macOS, Linux).

Follow the prompts in the installer to complete installation.

Verify Installation:

Open terminal/command prompt.

Run node -v to check the installed version.

Run npm -v to check the npm version.