CODEBOOSTERS TECH - NODE JS AND GRAPHQL HANDS ON TRAINING

Unit-I: Introduction to Node.js

Basics of Node.js

Imagine a **college canteen** that has only **one person taking orders**. Students keep coming in, placing orders, but instead of making them stand in a line, the canteen guy quickly **writes their order** on a pad and **keeps serving** others. The kitchen staff prepares the food in the background, and when it's ready, the canteen guy calls out the student's name.

Node.js works exactly like that.

- It’s **single-threaded —** one person handling multiple tasks.
- It's **non-blocking —** doesn’t wait for one task to finish before starting the next.
- Built on **Google Chrome’s V8 JavaScript engine**, Node.js is powerful and fast, especially for web applications.

Installation of Node.js

Installing Node.js is like **installing a new oven** in the canteen so that orders can be processed faster.

You download Node.js from its official website and install it. Along with it, you get **npm (Node Package Manager)**, which helps you bring in **ready-made ingredients** (packages) instead of preparing everything from scratch.

Working with Node Packages

Suppose your college organizes a **food fest** and you need to set up stalls.
Rather than cooking everything from scratch, you **buy readymade sauces, buns, drinks —** saving time and effort.

Similarly, in Node.js, **packages** are pre-built tools or libraries you can **install** and **use** to save time. Example:

- Need a web server? Use **Express** package.
- Need to connect to a database? Use **Mongoose** package.

Using Node Package Manager (npm)

npm is like **Amazon** for your food fest. You just search for "pizza base," "burger patty," click install, and it’s delivered.

In real terms,

- **npm install express** → installs the Express package.
- **npm list** → shows what’s installed.
- **npm update** → updates your packages.

Creating a Simple Node.js Application

Imagine you start a **simple lemonade stall** at the food fest.
You set up a table, a jug, and cups, and start serving.

In Node.js, your **first application** could be setting up a server that says, "**Welcome to the Food Fest!**" when someone visits your site.

Callbacks

Now, say you are managing two stalls — lemonade and popcorn.
You ask your friend to **inform you** once the popcorn machine is ready.
Until then, you continue serving lemonade.

This **inform-when-done** behavior is called a **callback** in Node.js.
You pass a function (like "Call me when done!") which runs after the task finishes.

Using Events

Think about your **college fest** again.
When the **chief guest arrives**, the **anchor** announces it immediately.

Events in Node.js are similar —

- An **event** occurs (like a file getting uploaded),
- You can **react** to it (like sending a "File Uploaded Successfully" message).

Listeners

Listeners are like **people who are waiting** to hear certain announcements.

For example:

- Listener 1: Waiting to hear when lunch is served.
- Listener 2: Waiting to hear when results are announced.

In Node.js, you **attach listeners** to events to perform actions when those events occur.

Timers

Imagine scheduling an **ice-cream eating competition** at exactly **4 PM**.

In Node.js, timers like **setTimeout** and **setInterval** help you:

- Do something **after a delay** (like announcing the ice-cream event after 5 minutes),
- Or **keep doing something repeatedly** (like playing music every 30 minutes).

Implementing HTTP Services in Node.js

Suppose you open an online ticket-booking service for your college fest movies.

When students visit your site:

- The server (Node.js) **receives requests** (for booking tickets),
- **Processes** them,
- And **sends back responses** (like "Booking Confirmed!").

In Node.js, you can create these **HTTP services** easily, allowing users to interact over the web.

Unit-II: Introduction to GraphQL

GraphQL vs REST API

Imagine ordering food at a movie theatre.

- With **REST API**, you must order **full meals —** even if you want just the drink.
- With **GraphQL**, you can **custom order exactly what you want —** “Only popcorn and a drink, no pizza!”

In REST:

- You might have different URLs for different things (movies, tickets, snacks).

In GraphQL:

- You hit **one URL** and **specify** exactly what you need in the request.

Key Differences:

- REST can be bulky.
- GraphQL is smart and precise — no under-fetching or over-fetching.

Setting up a GraphQL Server with Node.js

Setting up a GraphQL server is like setting up a **customized food counter** at a fest:

- One counter,
- One menu board,
- But people **request exactly what they want** and you **serve accordingly**.

Similarly, in Node.js, you set up a GraphQL server that listens to **specific queries** and responds **specifically**.

Understanding GraphQL Schema & Resolvers

Schema:
Imagine the **menu card** at your food stall.
It defines:

- What items are available (popcorn, coke, nachos).
- What ingredients each item has.

In GraphQL:

- The **schema** defines what data can be asked.

Resolvers:
Now, the **chef in the kitchen** makes the order based on the request.
Resolvers **fetch or process** the real data behind the scenes.

In GraphQL:

- Schema **describes** the types.
- Resolver **executes** the requests.

Query, Mutation, Validation

- **Query**:
  Like **asking the menu —** "Show me all movies playing today."

- **Mutation**:
  Like **placing an order —** "Book 2 tickets for 'Inception'".
- **Validation**:
  Before confirming your booking, they **check if seats are available**. Similarly, in GraphQL, input data is validated to make sure the request is correct.

Unit-III: Implementing GraphQL with Express Server

Introduction to GraphQL with Express

Imagine now you are upgrading from a single food counter to a **full restaurant**.
You set up a **professional ordering system**.

Express.js is like the **restaurant's service system —** handling all orders (requests) efficiently.

Combining Express with GraphQL allows you to create:

- A neat server,
- A single endpoint,
- Highly efficient communication.

Setting Up GraphQL with Express

It’s like setting up:

- A nice **front desk** (Express server),
- A **smart chef** (GraphQL).

You install necessary tools and configure Express to work smoothly with GraphQL.
The server listens to incoming GraphQL requests and responds dynamically.

Writing Queries and Mutations

Real-world Example:
A student visits the movie ticket counter:

- **Query**: "Show me movies available tonight."
- **Mutation**: "Book 2 tickets for 'The Batman'."

GraphQL understands whether you're **asking for information** (query) or **changing data** (mutation), and processes accordingly.

Integrating MongoDB/MySQL with GraphQL

Now you add a **database** (like a huge storage room) behind your restaurant.

- **MongoDB**: Like a **warehouse** where you can freely store anything (documents, recipes).
- **MySQL**: Like an **organized file cabinet** with strict folders and labels (structured data).

GraphQL connects to these databases:

- When a student asks for available movies,
- The server **fetches data** from MongoDB or MySQL,
- **Processes** it,
- And **delivers the response**.

📚 Conclusion

Through these fun real-world examples — canteens, food fests, movie counters, and college setups — you now understand how **Node.js** and **GraphQL** work in **real-life digital systems**:

- Fast, efficient event-driven operations (Node.js),
- Smart, flexible data handling (GraphQL).
