

# Week 11: Introduction to Express.js and API Routing

## Day 1: The Express Server Framework

### Theoretical Overview

Express.js is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications. It is the most popular framework for Node.js, designed to facilitate the rapid development of web applications and APIs. While Node.js provides the core http module, Express acts as a layer of "syntactic sugar," making it significantly easier to manage routes, handle requests, and integrate middleware.

### Setting Up an Express Project

To begin building the **Gadget API**, we first initialize a Node project and install Express using the Node Package Manager (NPM).

- **Installation:** Executing `npm install express` adds the library to the `node_modules` folder and updates the `package.json` file.
- **The App Instance:** The entry point of an Express application involves importing the module and invoking it to create an app object, which serves as the heart of the server.

## Day 2: Basic Routing and HTTP Methods

### Defining Routing Rules

Routing refers to determining how an application responds to a client request to a particular endpoint, which is a URI (or path) and a specific HTTP request method.

- **The Route Path:** This defines the endpoint (e.g., `/products` or `/users`) where the server listens for requests.

- **HTTP Methods:** Express provides methods that correspond to HTTP verbs:
  - **GET:** Used to retrieve data, such as a list of items from the **GadgetShop**.
  - **POST:** Used to submit new data, such as creating a new student record in the **LMS**.
  - **PUT/PATCH:** Used for updating existing records.
  - **DELETE:** Used to remove data from the database.

## Day 3: The Request and Response Objects

### The Request Object (req)

The req object represents the HTTP request and has properties for the request query string, parameters, body, and HTTP headers.

- **URL Parameters:** Express allows for dynamic routing by supplying URL parameters (e.g., /gadget/:id), which are accessed via req.params.
- **Query Strings:** Parameters passed after a ? in the URL (e.g., ?category=audio) are accessed via req.query.

### The Response Object (res)

The res object represents the HTTP response that an Express app sends when it receives an HTTP request.

- **res.send():** Sends a basic string or HTML response.
- **res.json():** Sends a JSON response, which is the standard format for the **Gadget API**.
- **res.status():** Sets the HTTP status code (e.g., 200 for success, 404 for not found).

## Day 4: Routing Logic and Execution Order

### Ordering of Routes

In Express, the order in which routes are defined is critical because the framework executes handlers in the sequence they appear in the code. If a generic route is placed before a specific one, the generic route will capture the request first.

### The Catch-All Route

A "Catch-All" route is defined at the very end of the routing file using a wildcard \*. This ensures that if a user requests an endpoint that does not exist in the **LMS** or **GadgetShop**, the server returns a 404 error rather than crashing or hanging.

## Day 5: Practical Implementation

This day is dedicated to building a full-example web service that integrates all concepts learned during the week.

- **Goal:** Create an Express server that handles multiple routes and returns structured data.
- **The Controller Pattern:** While the syllabus introduces basic routing, we begin organizing our **Gadget API** by moving logic into separate "Controller" functions to maintain clean, scalable code.
- **Serving Static Files:** We utilize `express.static` to serve product images for the **GadgetShop** frontend.