**Express.js**

**1. Introduction to Express.js**

**Definition**

Express.js is a minimal and flexible Node.js web application framework that provides a set of robust features for building web applications and APIs.

**Purpose**

Express.js simplifies the process of creating a web server in Node.js. It helps developers manage routes, handle requests and responses, and integrate middleware easily.

**Usage**

* Creating web applications
* Building RESTful APIs
* Handling HTTP requests and responses efficiently
* Managing routing in an organized way
* Handling routes, middleware, and requests efficiently.

**Description**

Express.js is built on top of Node.js and provides an abstraction layer that simplifies handling **HTTP requests, responses, and middleware**. Instead of writing low-level HTTP handling code, developers can use Express to **define routes, handle user input, and manage server-side logic** with minimal effort.

**How We Use It**

To use Express.js, we install it in a Node.js project and create a server that listens for incoming HTTP requests.

Install Express

Import Express into our project

Create an Express application

Define routes to handle client requests

Start the Express server to listen for requests

**Syntax**

To install Express.js:

>> npm install express

**Example:**

const express = require('express'); // Import Express

const app = express(); // Create an Express app

app.get('/', (req, res) => { // Define a route for home page

res.send('Welcome to Express.js'); // Send response

});

app.listen(3000, () => { // Start the server on port 3000

console.log('Server running on http://localhost:3000');

});

**2. Project Setup in Express.js**

**Definition**

Project setup in Express.js involves creating a new Node.js project, installing required dependencies, and organizing files for efficient development.

**Purpose**

Setting up a structured Express.js project ensures smooth development, easy scalability, and better code management.

**Usage**

* Initializes a new Node.js project
* Installs necessary dependencies (Express, middleware, etc.)
* Creates a proper folder structure for routes, views, and static files
* Prepares the project for development

**Description**

Before writing code in Express.js, we need to set up a project with the following steps:

1. Install Node.js (if not already installed)
2. Create a project folder
3. Initialize package.json
4. Install Express.js
5. Create an entry file (server.js or app.js)

**How We Use It**

**Step 1: Install Node.js**

Before using Express, we need Node.js installed.  
Check if Node.js is installed using:

>> node -v

If not installed, download it from <https://nodejs.org/>.

**Step 2: Create a Project Folder**

>> mkdir my-express-app

>> cd my-express-app

**Step 3: Initialize package.json**

>> npm init -y

This creates a package.json file that stores project details.

**Step 4: Install Express**

>> npm install express

This installs Express.js and saves it in node\_modules.

**4. Basic Routing in Express.js**

**Definition**

Routing in Express.js determines how an application responds to client requests for specific URLs (endpoints) using HTTP methods like GET, POST, PUT, and DELETE.

**Purpose**

Routing allows different parts of a web application to respond to different user requests. It helps structure the application efficiently, making it easier to handle multiple pages, forms, and APIs.

**Usage**

* Handling different user requests (Home, About, Contact pages)
* Managing API endpoints
* Directing users to different pages based on their requests

**Description**

Express routing allows us to:

1. Define routes for different URLs
2. Handle **GET, POST, PUT, DELETE** requests
3. Send responses like text, JSON, or HTML
4. Redirect users when necessary

**How We Use It**

To define routes, we:

1. Use app.get() for handling GET requests
2. Use app.post() for handling form submissions
3. Use app.put() for updating data
4. Use app.delete() for removing data

**5. Sending Data in Express.js**

**Definition**

Sending data in Express.js means responding to client requests with different types of data such as plain text, HTML, JSON, or files.

**Purpose**

When a user visits a website or an API endpoint, the server needs to send back some data, like:

* A webpage (HTML)
* Some text (plain text)
* Structured data (JSON)
* A file (PDF, image, etc.)

**Usage**

* Serving webpages
* Providing API responses
* Sending files for download

**Description**

There are multiple ways to send data in Express:

1. **Sending Text Data** – Useful for simple messages.
2. **Sending JSON Data** – Common for APIs and dynamic content.
3. **Sending HTML Content** – Used for serving web pages.
4. **Redirecting Users** – Used for guiding users to another route.

**6. Rendering HTML in Express.js**

**Definition**

Rendering HTML in Express.js means sending dynamic or static HTML pages as responses to client requests.

**Purpose**

* Helps serve web pages instead of just plain text or JSON.
* Allows dynamic content using templating engines like EJS, Handlebars, or Pug.
* Essential for full-stack web applications.

**Usage**

* Displaying web pages dynamically
* Creating templates with variables
* Making reusable UI components

**Description**

There are two main ways to render HTML in Express.js:

1. **Serving Static HTML Files** – Sending pre-written HTML files as responses.
2. **Using Template Engines** – Dynamically generating HTML using variables and logic.

**7. Routers in Express.js**

**Definition**

In Express.js, a router is a way to organize and manage different routes (URLs) separately, making the code modular and easy to maintain.

**Purpose**

* Helps break large Express applications into smaller, manageable parts.
* Allows separation of routes for better organization.
* Improves code readability and maintainability.

**Usage**

* Large applications with multiple routes.
* Keeping API routes separate from other logic.
* Grouping similar routes together (e.g., user-related routes, product-related routes).

**Description**

In Express.js, we create routers using express.Router(), then use them in the main app.

Steps:

1. **Create a separate router file** (userRoutes.js).
2. **Define routes inside that file**.
3. **Import and use the router in app.js**.

**8. Advanced Routing in Express.js**

**Definition**

Advanced Routing in Express.js allows handling **complex routes**, **multiple HTTP methods**, and **route parameters** efficiently. It includes:

1. **Route parameters** (/user/:id)
2. **Multiple HTTP methods on the same route**
3. **Route grouping**
4. **Route chaining**

**Purpose**

* Helps manage **dynamic** and **parameterized** routes.
* Reduces **code duplication** by grouping related routes.
* Improves **API design** by supporting multiple HTTP methods per route.

**Usage**

* APIs handling **user profiles** (/users/:id)
* **E-commerce** routes for products/orders
* **Filtering/sorting** routes (/products?category=mobiles)

**Description**

Advanced routing involves:

1. **Route Parameters** – Handling dynamic values.
2. **Multiple HTTP Methods** – GET, POST, PUT, DELETE.
3. **Route Grouping** – Keeping related routes together.
4. **Middleware for specific routes** – Processing before sending responses.

**9. Middleware in Express.js**

**Definition**

Middleware in Express.js is a function that runs **between the request and response cycle**. It can modify the request, response, or even stop the request from reaching the final route handler.

**Purpose**

* Logging requests
* Authentication & Authorization
* Parsing incoming data (JSON, forms, etc.)
* Error handling

**Usage**

* **Global Middleware:** Runs for all requests (e.g., logging, authentication).
* **Route-Specific Middleware:** Runs only for specific routes (e.g., checking if a user is logged in before accessing a page).
* **Built-in Middleware:** Express provides middleware like express.json() for parsing JSON.
* **Third-party Middleware:** Middleware from external libraries like cors for enabling Cross-Origin Resource Sharing.

**Description**

Middleware functions are of different types:

1. Application-Level Middleware – Runs on every request.
2. Router-Level Middleware – Runs only on specific routes.
3. Built-in Middleware – Express.js pre-defined functions (e.g., express.json()).
4. Error-Handling Middleware – Catches and processes errors.

**10. Rendering Static Files in Express.js**

**Definition**

Rendering static files in Express.js means serving files like HTML, CSS, images, JavaScript, fonts, or PDFs from a directory so users can access them via a browser.

**Purpose**

* Allows websites to serve front-end files (like CSS, JS).
* Used for loading images, styles, or scripts in web applications.
* Helps in serving static web pages (e.g., documentation, company profiles).

**Usage**

* Serving HTML, CSS, and JavaScript files.
* Displaying images (logos, banners).
* Hosting downloadable files (PDFs, documents).

**Description**

Express provides a built-in middleware express.static() to serve static files.

* We **place all static files in a folder** (e.g., public/).
* Express **automatically delivers** files when requested.

**11. Parsing Form/JSON Data in Express.js**

**Definition**

Parsing Form/JSON data in Express.js means **extracting data sent by users** from **HTML forms** or **JSON requests** and making it accessible in our server.

**Purpose**

* Enables Express.js to **read form inputs** from users.
* Allows Express.js to **handle JSON data** sent from the client.
* Used in **login forms, contact forms, or API requests**.

**Usage**

* **Form Data Parsing** → When users fill and submit an HTML form.
* **JSON Data Parsing** → When an API sends data in JSON format.

**Description**

Express provides built-in middleware to parse:

1. **Form Data** – express.urlencoded({ extended: true })
2. **JSON Data** – express.json()

**12. Parsing Query Parameters in Express.js**

**Definition**

Query parameters are **key-value pairs** sent in the URL after a ?. Express.js allows us to **extract and use** these parameters in our routes.

**Purpose**

* Used for **filtering/searching data** (e.g., searching products on Amazon).
* Helps **pass small data in URLs** (e.g., tracking user referrals).
* Commonly seen in **Google searches, YouTube links, and pagination**.

**Usage**

* http://example.com/products?category=mobiles&brand=apple
  + Here, category and brand are **query parameters**.

**Description**

Query parameters are sent in the **URL after a ?** and are separated by &:

bash

CopyEdit

http://example.com/products?category=electronics&sort=price\_high\_to\_low

* category=electronics → Shows only electronics.
* sort=price\_high\_to\_low → Sorts results from highest to lowest price.