



# Node and Express

SC 353 Short Course on Intro to NodeJS & MongoDB

Reuben Devanesan  
Kanishk Singhal  
Ananthu J P

# What is Express.js?



Express.js is a minimalist web application framework for Node.js, designed to make building web applications and APIs with Node.js easier and more organized. It provides a robust set of features for web and mobile applications, such as **routing**, **middleware support**, **templating engines**, and much more.

**Brief history and background :** Express.js was created by **TJ Holowaychuk** and initially released in **2010**. It quickly gained popularity due to its **simplicity and flexibility**. Over the years, it has become one of the most widely used frameworks for building web applications with Node.js.

## Features

- **Lightweight and flexible:** Express.js is lightweight, allowing developers to structure their applications as they see fit.
- **Robust routing:** Express.js provides a powerful routing mechanism, making it easy to define routes for handling different HTTP requests.
- **Middleware support:** Express.js has built-in middleware support, allowing developers to easily extend the functionality of their applications.
- **Wide community adoption:** Express.js has a large and active community, with plenty of resources, tutorials, and third-party middleware available

# Getting Started with Express.js:




1. Setup node js app and install express.



```
npm init  
npm install express
```

2. Import express module and create an instance:



```
var express = require('express')  
var app = express()
```



## Basic routing

**Routing** in web applications refers to the process of mapping **HTTP requests** to **specific endpoints** (URLs) and defining how the **server should respond** to each request.

Each route can have one or more handler functions, which are executed when the route is matched.

Syntax :



```
app.METHOD(PATH, HANDLER)
```

- **app** is just an instance of Express.js. You can use any variable of your choice .
- **METHOD** is an HTTP request method such as get, set, put, delete, etc.
- **PATH** is the route to the server for a specific webpage
- **HANDLER** is the callback function that is executed when the matching route is found.

# Handler function

*A code block that is responsible for handling incoming requests and providing appropriate responses to an API*

```
app.get("/", (req, res) => {  
  res.send("Hello World");  
});
```

## 1. Response

`res.send()` - forwards any data passed as an argument to the client-side. The method can take a **string**, **array**, and an **object** as an argument.

`res.download()` - function transfers the file at the path as an 'attachment'.

`res.render()` - used to render a view and sends the rendered HTML string to the client.

`res.status()` - function sets the HTTP status for the response. [\[ref\]](#)

# Handler function



```
app.get("/:id", (req, res) => {  
  if (req.params.id > ArrayOfFruits.length) {  
    throw new Error(511);  
  } else {  
    res.send(req.params.id + ":" + ArrayOfFruits[req.params.id]);  
  }  
});
```

## 2. Request

**req.params()** - object containing properties mapped to the named route “**parameters**”. For example, if you have the route `/student/:id`, then the “`id`” property is available as `req.params.id`.

**req.query()** - allows you to access the query parameters from the URL of an incoming HTTP request. Query parameters are key-value pairs included in the URL after the “?” symbol, and they are separated by “&” symbols.

**req.header()** - A section of your requests that's used to transfer metadata about the request and the desired response. [e.g: host, user-agent, custom tokens etc.]

**req.body()** - A request body is data sent by the client to your API. A response body is the data your API sends to the client. But clients don't necessarily need to send request bodies all the time.

# express.Router



Use the `express.Router` class to create **modular, mountable** route handlers. A Router instance is a complete middleware and routing system; for this reason, it is often referred to as a “mini-app”.



```
const express = require('express')
const router = express.Router()
app.use('/route', router)
```

# Middleware

Middleware functions are functions that have access to the **request object** (req), the **response object** (res), and the **next middleware function** in the application's **request-response cycle**. The next middleware function is commonly denoted by a variable named next.

Middleware functions can perform the following tasks:

- Execute any code.
- Make changes to the request and the response objects.
- End the request-response cycle.
- Call the next middleware function in the stack.

**Types:** Application-level middleware, Router-level middleware, Error-handling middleware, Built-in middleware, Third-party middleware, Custom middleware

**Example : Body-parser :** Parse incoming request bodies in a middleware before your handlers, available under the req.body property.

JSON Parse :

```
app.use(express.json());
```



# Error Handler Middleware

Express catches and processes errors that occur both synchronously and asynchronously. Express comes with a default error handler so you don't need to write your own to get started.

```
throw new Error("Server Error");
```

Custom Error handler :

```
const errorHandler = (err, req, res, next) => {  
  console.log({title:err.message, stackTrace: err.trace})  
};
```