Lesson:

Express part 01

PW SKILLS

Topics

- What is FrameWork?
- What is Express.js
- Understanding the request-response cycle in Express.js
- Challenges in using only node.js
- Why we use express
- Express setup

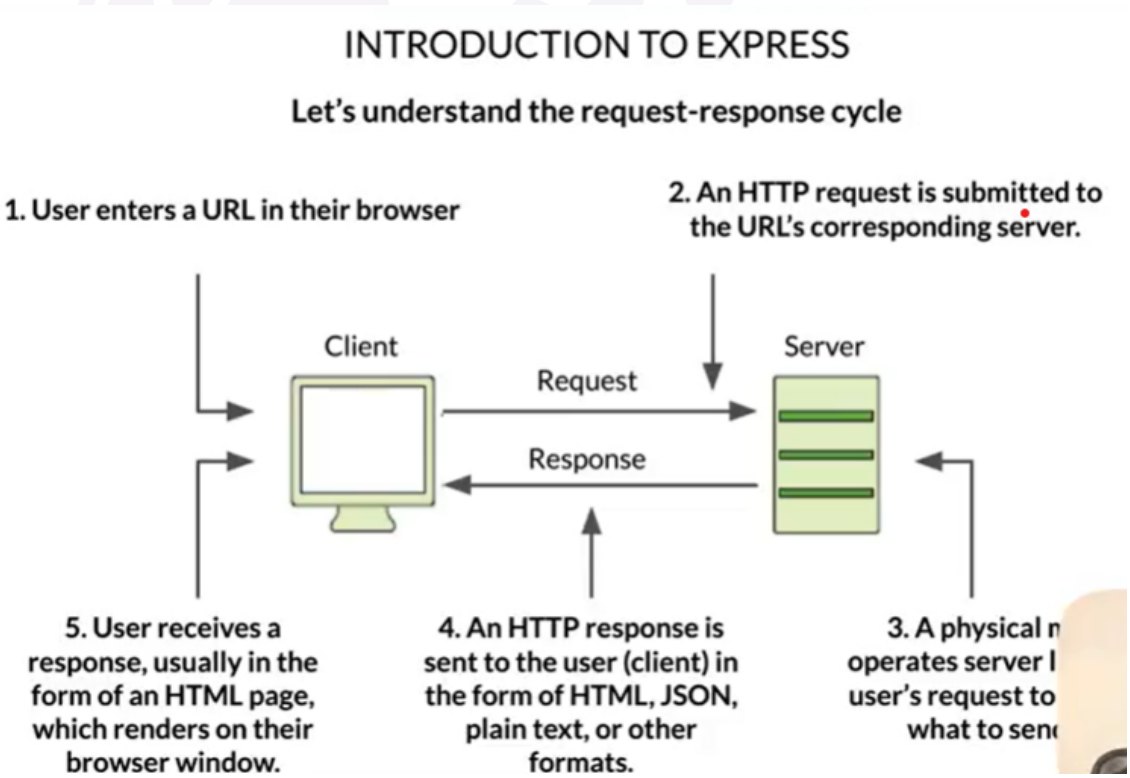
What is FrameWork?

A framework is a collection of tools and libraries that developers can use to build software applications more efficiently. It provides a structured way to organize code, making it easier to develop, maintain, and scale applications. Frameworks often include pre-written code for common tasks, such as handling user input, managing databases, and rendering web pages. This pre-written code acts as a foundation, allowing developers to focus on the unique aspects of their applications rather than the basic building blocks.

What is Express ?

Express is a web application framework for Node.js, a popular JavaScript runtime. It simplifies the process of building web applications and APIs by providing a set of tools and conventions. Express allows developers to create robust and scalable web applications with minimal effort. It is known for its simplicity, flexibility, and ease of use.

Understanding the request-response cycle in Express.js



Understanding the request-response cycle in Express.js involves recognizing how a user's action (entering a URL in their browser) triggers a series of events that ultimately result in the user receiving a response, typically in the form of an HTML page. Here's a breakdown of the points you've mentioned, weighing their significance in the cycle:

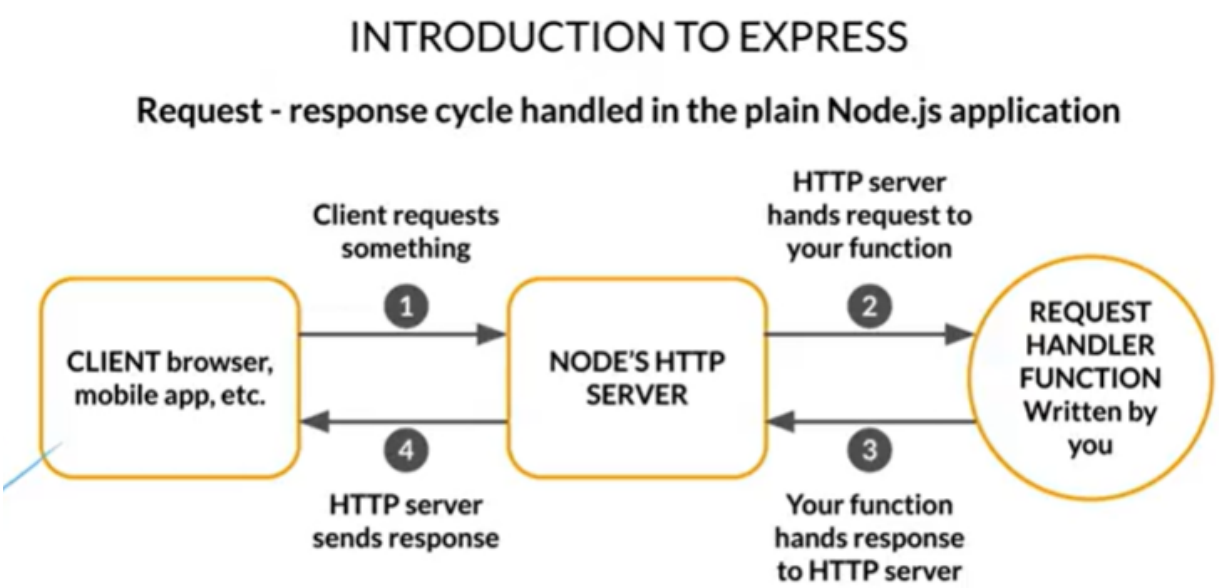
User enters a URL in their browser: This is the starting point of the request-response cycle. The user's action initiates the process by specifying the resource they want to access.

An HTTP request is submitted to the URL's corresponding server: Once the user enters a URL, the browser sends an HTTP request to the server that hosts the website. This request includes information about what the user wants to do (e.g., view a page, submit data).

A physical node (server) operates server logic to determine what to send: The server receives the HTTP request and processes it. This involves executing server-side logic, which might involve querying a database, performing calculations, or interacting with other services. The server then decides what data or resource to send back to the client.

An HTTP response is sent to the user (client) in the form of HTML, JSON, plain text, or other formats: After processing the request, the server sends back an HTTP response. This response contains the requested data or resource, which could be in various formats such as HTML for web pages, JSON for data exchange, or plain text for simple messages.

Users receive a response, usually in the form of an HTML page, which renders on their browser window: The client (user's browser) receives the HTTP response from the server. If the response contains HTML, the browser renders the HTML into a visual webpage that the user can interact with. This completes the request-response cycle.



Challenges in using only node.js

As the number of types of client requests increases, handling those gets complex:

- Dealing with various user actions on a website becomes complicated without a framework like Express. Managing different types of client requests becomes like juggling multiple tasks without a clear structure, leading to potential confusion and errors.

Many of the commonly used features, such as reusable HTML templates, require a lot of code manipulation:

- Working with reusable HTML templates can be labor-intensive without Express. Manual manipulation of code for creating or updating pages becomes time-consuming and error-prone. Express simplifies this process with tools and conventions.

There is a lot of boilerplate code:

- Starting a new web project without a framework results in writing a lot of repetitive boilerplate code. Express reduces this burden by providing built-in structures and conventions, allowing developers to focus more on building unique features rather than reinventing the foundational code.

Why we use express

Faster Server-Side Development:

- Express makes it quicker to develop the server side of your application. It speeds up the process, so you can build things faster.

Pluggable Middleware:

- Think of middleware as helpers for your server. Express lets you easily plug in these helpers to add extra features or functions to your application.

Support for Templating Engines:

- Templating engines are like blueprints for building web pages. Express supports these blueprints, making it easier to create and manage dynamic web content.

Clean Routing Support:

- Routing is like guiding traffic in your application. Express helps you keep it clean and organized, making it easy to direct users to the right places.

Debugging Support:

- When things go wrong, debugging helps find and fix issues. Express provides support for debugging, making it easier to identify and solve problems in your code.

Express setup

Getting started with Express.js is relatively straightforward. Here's a step-by-step guide on setting up Express.js:

Prerequisites:

Make sure you have Node.js installed on your machine. You can download it from Node.js official website.
https://nodejs.org/en

Create a New Project:

- Open your terminal or command prompt.
- Create a new project folder: mkdir Express
- Move into the project directory: cd Express

Initialize a Node.js Project:

Run the following command to initialize a new Node.js project. This will create a package.json file:

```
Unset
npm init
```

Package.json file:

```
{
  "name": "express",
  "version": "1.0.0",
  "description": "We are learning express",
  "main": "index.js",
  ▷ Debug
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "express": "^4.18.2"
  }
}
```

Install Express:

Install Express as a dependency for your project:

```
Unset
npm install express
```

Create an Express App:

- Create a file (e.g., app.js or index.js) to write your Express application code.
- Open the file in a text editor.

```
Unset
const express = require('express');
const app = express();

/**
* see a req- res cycle
**/
app.get('/', (req, res) => {
  res.send('Hello from the Express server');
});

/**
* starting express server
**/
app.listen(8000, () => {
  console.log(`Server got started`);
});
```

Run Your Express App:

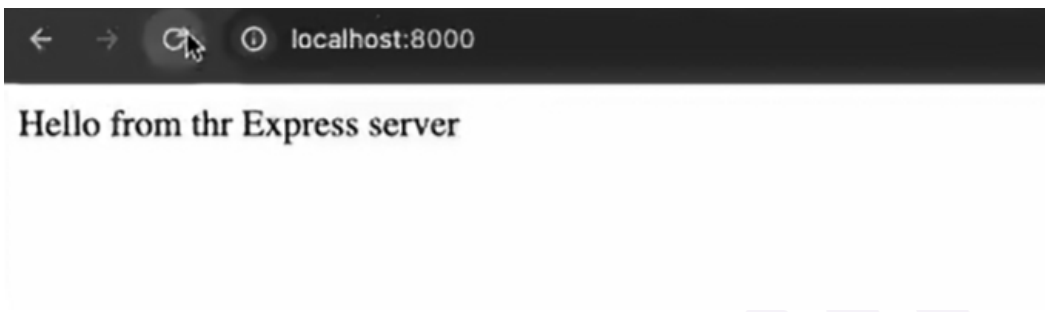
- Save your changes in the index.js file.
- In the terminal, run the following command to start your Express app

```
Unset
node app.js
```

Test Your Express App:

- Open your web browser and go to **http://localhost:8000**. You should see the message "Hello from the Express server".

Browser output



Congratulations! You've set up a basic Express.js application. From here, you can gradually explore more advanced features, middleware, routing, and other functionalities that Express.js offers.

PW SKILLS

THANK YOU !