SWE 4538: SERVER PROGRAMMING LAB

LAB_01: Web Development & Basics of Nodejs

PREPARED BY :

SHOHEL AHMED || ASSISTANT PROFESSOR
a.shohel@iut-dhaka.edu

FARZANA TABASSUM || LECTURER
farzana@iut-dhaka.edu

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
ISLAMIC UNIVERSITY OF TECHNOLOGY

SEPTEMBER 10, 2024

Contents

1 Web Development

Web development involves creating websites or web applications that users can interact with through browsers. It encompasses both frontend (client-side) and backend (server-side) development. In this lab, we'll learn about server-side development in detail.

1.1 Basic components of web applications

A web application is a program that runs on a server and is rendered by a client browser, using the internet to access all the resources of that application. It usually can be easily broken down into three parts: Client, Server, & Database.

- **Client:** The user interacts with the front-end part of a web application. The front-end is usually developed using languages like HTML and CSS styles, along with extensive usage of JavaScript-based frameworks like ReactJS and Angular, which help with application design.
- **Server:** The server is responsible for taking the client requests, performing the required tasks, and sending responses back to the clients. It acts as a middleware between the front-end and stored data to enable operations on the data by a client. Node.js, PHP, and Java are the most popular technologies in use to develop and maintain a web server.
- **Database:** The database stores the data for a web application. The data can be created, updated, and deleted whenever the client requests. MySQL and MongoDB are among the most popular databases used to store data for web applications.

1.2 How the Browser Interacts With the Servers?

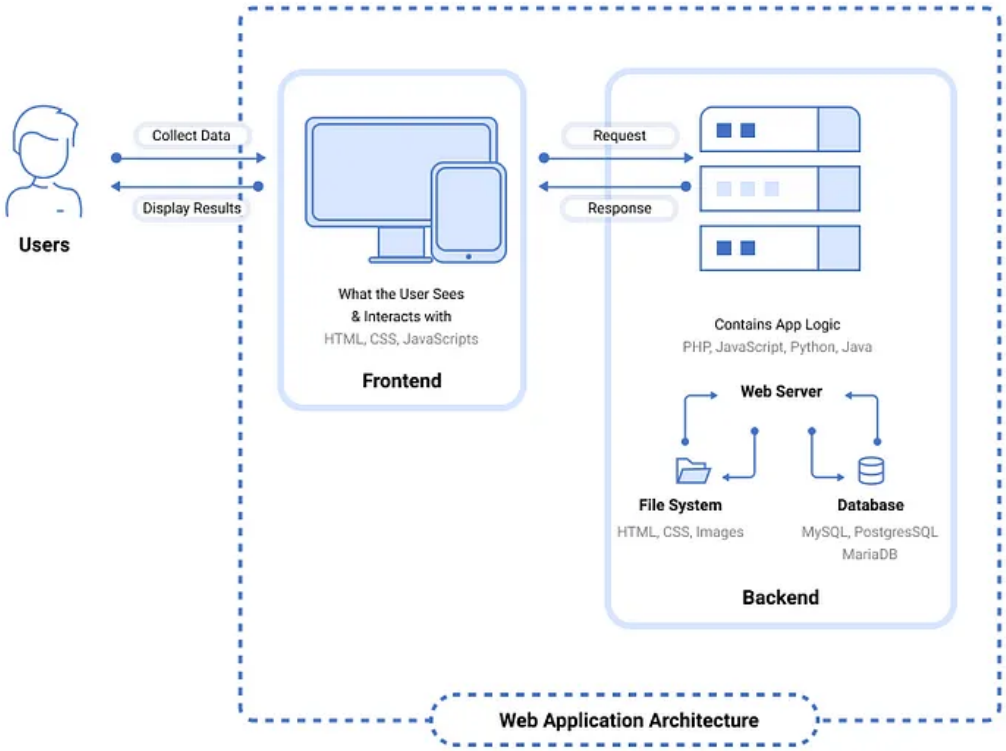

Figure 1: Client Server Architecture

- User enters the URL(Uniform Resource Locator) of the website or file. The Browser then requests the DNS(DOMAIN NAME SYSTEM) Server.

- DNS Server lookup for the address of the WEB Server.
- The DNS Server responds with the IP address of the WEB Server.
- The Browser sends over an HTTP/HTTPS request to the WEB Server's IP (provided by the DNS server).
- The Server sends over the necessary files for the website.
- The Browser then renders the files and the website is displayed. This rendering is done with the help of DOM (Document Object Model) interpreter, CSS interpreter, and JS Engine collectively known as the JIT or (Just in Time) Compilers.

2 Node.js: An Overview

Node.js is an open-source, cross-platform JavaScript runtime environment built on the Chrome V8 JavaScript engine. It enables developers to execute JavaScript code outside of web browsers, making it suitable for server-side programming. Node.js empowers developers to create a variety of applications, including web servers, APIs, command-line tools, and more. Key Features of Node.js are as follows:

1. **Non-Blocking and Asynchronous:** Node.js follows a non-blocking and asynchronous approach. Its event-driven, single-threaded architecture efficiently handles concurrent requests without blocking other tasks. This is particularly effective for I/O-intensive operations.
2. **V8 JavaScript Engine:** Node.js leverages the V8 JavaScript engine, shared with the Google Chrome browser.
3. **NPM (Node Package Manager):** NPM is the default package manager for Node.js. It simplifies package installation, management, and updates.
4. **Module System:** Node.js employs a module system that organizes code into reusable components.
5. **Cross-Platform:** Node.js is cross-platform, running on diverse operating systems such as Windows, macOS, and Linux.

3 Introduction to Express

Express.js, commonly referred to as Express, is a minimal and flexible web application framework for Node.js. It provides a robust set of features and tools for building web and mobile applications, APIs (Application Programming Interfaces), and other server-side applications. Express is designed to make server-side development with Node.js more streamlined, organized, and efficient.

3.1 Key Features of Express.js

- **Routing System:** Developers can define routes to handle different HTTP requests (GET, POST, PUT, DELETE, etc.) for various URLs. This simplifies mapping URLs to specific functionalities.
- **Middleware Support:** Middleware are functions that process incoming requests before they reach the route handler. Express enables developers to utilize middleware for tasks such as authentication, data validation, logging, and error handling. Middleware functions execute in the order they're defined.

- **HTTP Utility Methods and Middleware:** Express includes built-in HTTP utility methods and middleware that streamline tasks like parsing request bodies, managing cookies, and serving static files.
- **Error Handling:** Express offers mechanisms to handle both synchronous and asynchronous errors consistently. Custom error handling middleware can be implemented to capture and respond to errors.
- **RESTful API Development:** Express is frequently used to construct RESTful APIs due to its routing system and middleware support. It simplifies handling various HTTP methods and CRUD operations.

4 Environment Setup for Web Development with Express.js

4.1 Your First Express App

Setting up the environment for web development with Express.js involves installing the necessary tools and packages. Follow these steps to get started:

1. **Install Node.js and npm:** Express.js is built on top of Node.js, so you need to install Node.js and its package manager, npm. Visit https://nodejs.org/en/download and download the appropriate installer for your operating system. Follow the installation instructions to complete the setup.
2. **Create a Project Directory:** Open a terminal or command prompt and navigate to the location where you want to create your project directory. Create a new directory for your Express.js project using the following command:

```
mkdir my-express-app  // create a new directory/folder
cd my-express-app     // navigate to that directory
```

3. **Initialize a Node.js Project:** Inside your project directory, initialize a new Node.js project using the following command:

```
npm init
```

You'll then be guided through a series of prompts, such as:

```
name: (my-node-project)
version: (1.0.0)
description: A sample Node.js project
entry point: (index.js)
... (more prompts)
```

These questions cover aspects like project name, version, description, entry point, repository URL, license, and more. For each question, you can provide your own values or simply press Enter to accept the default values enclosed in parentheses. Default values are often based on your previous inputs.

Initializing a Node.js project using the `npm init` command is a fundamental step in setting up your project environment. This process involves creating a **package.json** file, which serves as the metadata and configuration file for your project. Once the `package.json` file is generated, you can open it in a text editor to review and modify its contents. You can add or update information as needed, such as dependencies, scripts, and more.

4. **Install Express:** Install the Express.js framework as a project dependency using npm. In your terminal, run the following command:

```
npm install express
```

The `npm` command is the Node Package Manager, responsible for managing packages and dependencies for Node.js projects. They are stored in the "node_modules" directory within your project's root directory. npm updates the `package.json` file of your project to include an entry for the "express" package as a dependency. It also generates a `package-lock.json` file. This file provides a detailed, version-specific record of the installed packages and their dependencies. It ensures that future installations will use the exact same versions, preventing unintended changes due to updates. Once the installation process is complete, you can start using the "express" package in your Node.js project.

5. **Create Your Express Application:** Now that your environment is set up, you can start building your Express application. Create a new JavaScript file (e.g., `app.js`) in your project directory. This will be the main file of your application.

6. **Import Express:** In `app.js`, start by importing the Express module at the beginning of the file:

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

The first line of code imports the Express module into your Node.js application. In Node.js, the `require` function is used to include external modules or files in your program. The second line of code initializes an instance of the Express application by invoking the `express()` function. After these two lines of code, the `app` variable holds your Express application instance. You can now use this `app` object to set up various aspects of your web application, such as defining routes to handle different HTTP requests, applying middleware, serving static files, and more.

7. **Start the Server:** Finally, start your Express server by listening on a specific port. Add the following code at the end of `app.js`:

```
const PORT = 3000;
app.listen(PORT, () => {
    console.log('Server is running on port ${PORT}');
});
```

Replace 3000 with the desired port number for your application. The `app.listen()` method is used to start the Express application and make it listen for incoming HTTP requests. It binds the application to a specific port on the host machine, allowing it to handle requests and responses over that port. The `console.log()` function is a built-in Node.js function that outputs a message to the console.

Congratulations, you've successfully set up your environment for web development with Express.js! You can now begin building your web applications, APIs, and more using the power of Express.

4.2 Arrow Functions in JavaScript

Arrow functions, introduced in ECMAScript 6 (ES6), provide a concise and efficient way to write functions in JavaScript. They offer a more compact syntax compared to traditional function expressions and come with unique behavior and considerations. Here's a detailed explanation of arrow functions:

4.2.1 Syntax

The syntax of an arrow function looks like this:

```
(parameters) => {
    \\ statements
}
```

Example:

```
// Regular function expression
const add = function(x, y) {
    return x + y;
};

// Arrow function (single expression)
const subtract = (x, y) => x - y;

// Arrow function with multiple statements
const greet = name => {
    const message = `Hello, ${name}!`;
    console.log(message);
};
```

5 Hello World with Express

We will create another web application with Express. When accessed, it will send the response 'Hello, World!' back to the client. This is a basic example of defining routes and responding to client requests in an Express.js application.

5.1 Creating an Express Application

Create a file named `app.js` in your project directory. This will be your main application file.

5.2 Importing Packages

At the top of `app.js`, import the Express module:

```
const express = require('express'); //Import the express module
const app = express(); //Initialize the express app
```

5.3 Defining a Route

Define a basic route that responds with "Hello, World!" when a user accesses the root URL:

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

Let's break down the code snippet `app.get('/', (req, res) => { ... });` in detail:

- `app.get()` Method: Is used to define routes for handling specific HTTP methods. In this case, we're focusing on GET requests.
- Route Path: In the code snippet, `'/'` is the route path. It specifies the URL path at which the route will be triggered. In this example, `'/'` represents the root path of the application.
- Callback Function: The callback function `(req, res) => { ... }` is executed when a client makes a GET request to the specified route. The callback function takes two parameters: `req` (request) and `res` (response).
- `req` - Request Object: The `req` object represents the incoming HTTP request from the client. It contains various properties and information about the request, such as headers, query parameters, and the URL.
- `res` - Response Object: The `res` object represents the outgoing HTTP response that will be sent back to the client. It provides methods to customize the response, set headers, and send data.
- `res.send()` Method: The `res.send()` method is used to send a response back to the client. In this case, it sends the string `'Hello, World!'` as the response body. It can be a other types of file as well. The client's browser will display the reply upon accessing the route.

5.4 Static Files

To serve static files such as images, CSS files, and JavaScript files, use the `express.static` built-in middleware function in Express.

```
express.static(root, [options])
```

The `root` argument specifies the root directory from which to serve static assets. For example, use the following code to serve images, CSS files, and JavaScript files in a directory named **public**:

```
app.use(express.static('public'))
```

5.5 Starting the Server

Start the Express server to listen on a specific port:

```
const PORT = 3000;
app.listen(PORT, () => {
    console.log(`Server is running on port ${PORT}`);
});
```