Backend Programming with Express: Class 1

Tasnim Ahmed

August 16, 2023

1 Introduction to Web Development and Express

1.1 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.

1.2 Benefits of Backend Frameworks

The backend handles server-side logic, data storage, and communication with databases and other services. Utilizing a backend framework offers several advantages:

- **Efficiency:** Backend frameworks provide a structured foundation for building server-side applications. They offer pre-built components, libraries, and tools that streamline development, reducing the time and effort required to create complex functionalities.
- **Modularity:** Frameworks promote modular coding practices, allowing developers to break down applications into smaller, manageable components. This enhances code organization, reusability, and maintenance, making it easier to collaborate on larger projects.
- **Security:** Many backend frameworks come with built-in security features, such as protection against common vulnerabilities like SQL injection, cross-site scripting (XSS), and cross-site request forgery (CSRF). These features help safeguard applications and user data.
- **Scalability:** Backend frameworks often incorporate best practices for handling high levels of traffic and user requests. This scalability ensures that applications can grow and accommodate increased demand without sacrificing performance.
- **Consistency:** By following the conventions and patterns established by a backend framework, developers can maintain a consistent coding style throughout the application. This consistency improves code readability and eases the onboarding of new team members.

1.3 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. V8 compiles JavaScript code into machine code, resulting in high-performance execution and optimization.
3. **NPM (Node Package Manager):** NPM is the default package manager for Node.js. It offers a vast ecosystem of open-source libraries and modules that streamline application development. NPM simplifies package installation, management, and updates.

4. **Module System:** Node.js employs a module system that organizes code into reusable components. Each file is treated as a module, allowing for import and use across different parts of the application. This enhances code organization and reusability.

5. **Event-Driven Architecture:** Node.js employs an event-driven architecture for handling asynchronous operations. An event loop continuously monitors events and triggers corresponding callbacks. This architecture facilitates handling numerous connections concurrently.

6. **Cross-Platform:** Node.js is cross-platform, running on diverse operating systems such as Windows, macOS, and Linux. This flexibility enables code to be written once and deployed across various environments.

7. **Community and Ecosystem:** Node.js boasts a large and active developer community, contributing to a rich ecosystem of libraries, frameworks, and tools. This collaborative approach leads to continuous improvements, updates, and abundant learning resources.

8. **Web Servers and APIs:** Node.js is commonly used for building web servers and APIs. Frameworks like Express.js facilitate the creation of server-side applications that handle requests, route traffic, and interact with databases.

9. **Command-Line Tools:** Node.js is also employed to develop command-line tools and scripts. Its simplicity and available libraries make it a powerful choice for automating tasks and creating custom command-line interfaces.

1.4 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.

1.5 Key Features of Express.js

- **Minimalistic and Unopinionated:** Express adopts a minimalist approach that provides essential structure while allowing developers flexibility in designing project architectures. It doesn't enforce rigid patterns, allowing for diverse project structures.

- **Routing System:** Express offers a powerful 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.

- **Template Engines:** While not providing a specific template engine, Express seamlessly integrates with engines like EJS, Pug, and Handlebars. These engines facilitate dynamic HTML content generation.

- **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.

- **Extensibility and Middleware Ecosystem:** Express's simplicity doesn't hinder its extensibility. Developers can incorporate third-party middleware and libraries to enhance its capabilities. The Node.js ecosystem offers a wide array of compatible middleware.

- **Community and Documentation:** A vibrant developer community contributes to a wealth of tutorials, examples, and resources. The official documentation is comprehensive and well-maintained, accommodating beginners and experienced developers alike.
- **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.
- **Web Application Development:** In addition to APIs, Express is suitable for building web applications with server-side rendering. Integration of template engines and management of routes based on user interactions enable dynamic HTML page rendering.

2 Environment Setup for Web Development with Express.js

2.1 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 the official Node.js website 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
   cd my-express-app
   ```

   In the context provided, the commands `mkdir` and `cd` are used in a terminal or command prompt to manage directories (folders) within a file system. The `mkdir` command stands for "make directory". It is utilized to create a new directory (folder) in the specified location. In the example given, the command `mkdir my-express-app` creates a directory named "my-express-app" in the current location, which is the directory where you are currently working in the terminal. The `cd` command, short for "change directory", is used to navigate into a directory. In the provided example, the command `cd my-express-app` changes the current working directory to "my-express-app." This means that any subsequent commands executed will operate within the "my-express-app" 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. When you run `npm install express`, npm searches the registry for the latest version of the "express" package. Once npm identifies the package and version to install (or the version specified in the command, if any), it begins downloading the "express" package and its dependencies from the registry. The downloaded package files 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. The version of "express" installed is specified in the dependencies section of the `package.json` file. If "express" has its own dependencies, npm will install those dependencies recursively as well, creating a dependency tree. Alongside the `package.json` file, npm 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.

2.2 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:

2.2.1 Syntax

The syntax of an arrow function looks like this:

```
(parameters) => expression
```

For functions with multiple statements, the syntax is:

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

2.2.2 Concise Syntax

Arrow functions are particularly useful for writing short and concise functions. When the function body consists of a single expression, you can omit the curly braces {} and the `return` keyword. The result of the expression is automatically returned.

```
const multiply = (x, y) => x * y;
```

2.2.3 Use Cases

Arrow functions are particularly suitable for short, simple functions and scenarios where you want to maintain the value of `this` from an outer scope. However, they might not be appropriate for complex functions with multiple statements. Here are some examples to illustrate arrow functions:

```
// 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);
};
```

3 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.

3.1 Creating an Express Application

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

3.2 Importing Express

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

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

3.3 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!');
});
```

In an Express.js application, the `app.get()` method is used to define routes that handle HTTP GET requests. Let's break down the code snippet `app.get('/', (req, res) => { ... });` in detail:

- `app.get()` Method: The `app.get()` method is part of the Express.js framework and 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. The client's browser will display this string upon accessing the route.
- HTTP GET Request: The provided code handles HTTP GET requests to the root path of the application. When a client accesses the root URL, the `app.get()` route will be triggered, and the defined callback function will execute, responding with the string `'Hello, World!'`.
- Other HTTP Methods: Express.js also provides methods like `app.post()`, `app.put()`, and `app.delete()` to handle POST, PUT, and DELETE requests, respectively. These methods follow a similar structure for defining routes.

3.4 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}`);
});
```

4 Homework Assignment

1. Review the class materials on web development and Express.
2. Explore the Express.js documentation to understand routing concepts.
3. Modify the "Hello World" application to create new routes and responses.
4. Modify the "Hello World" application to send an `HTML` file back to the client.