ExpressJS - Web App Framework

What is it?

- ExpressJS, also known as Express, is a NodeJS backend web application framework used to build web applications and Application Programming Interfaces (APIs), which allow applications to access data, written in JavaScript, a programming/scripting language.

Use in modern web development

- Aids in the simplification of backend development

Where would you see it in real life?

- ExpressJS is used to design APIs, create real-time applications (dashboards, chatbots) as well as single-page HTML websites, sending API endpoints to the front end.

Basic Concepts and Terminology in ExpressJS

ExpressJS revolves around four core concepts:

1. **Routing** - defines how your application responds to different HTTP requests (GET, POST, PUT, DELETE, etc.) to specific URL endpoints; think of routing as the menu for a pizza restaurant where you choose what you want with varying customizations

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

- **Route** - A URL path with a specific HTTP method (GET, POST, PUT, DELETE)

- **Route Handler** - Function that executes when route is matched

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

HTTP Methods

- GET → Fetch data.

- POST → Submit data.
- PUT/PATCH → Update data.
- DELETE → Remove data.

2. **Middleware** - Functions that have access to the request, response, and next middleware in your application's request-response cycle; think of middleware as the person taking your order, the kitchen staff

3. **Request/Response objects** - Represent the HTTP requests and responses with different properties for requests (req.params, req.query, req.body, etc.) and methods for responses (res.json(), res.status(), res.send()); Requests are your specific order and response objects are the finished projects

4. **Express App** - The main object that configures routes, middleware, and server logic; This is the restaurant itself; enjoy your pizza!

   This is an example of the creation of an Express app

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

Core Syntax and Examples in ExpressJS

Below is an example of a basic server setup in ExpressJS. This block of code essentially loads the Express library to create an app, then prepares to parse through JSON data, listens for a GET request, and then begins running the server.

```
const express = require("express"); // Imports Express library
const app = express(); // Creates Express app

// Middleware
app.use(express.json()); // Used to parse JSON data

// Routing
// This line of code is used to get data from a route and send a message
once done successfully
app.get("/", (req, res) => res.send("Home Page"));
```

```
// Starts server once done
app.listen(3000, () => console.log("Server running on port 3000"));
```

Below is a more in-depth example of routing. The GET request is essentially calling to retrieve the user ID, attempting to obtain it from the URL (/api/users/:id). When the ID is received, it sends a status 200 to confirm that it was sent. The same is done in a POST request, once it gets the data, it sends a status to confirm it was created with a status 201.

```
// GET request handler
app.get('/api/users/:id', (req, res) => {
 const userId = req.params.id;
 res.status(200).json({ message: `User ID: ${userId}` });
});

// POST request handler
app.post('/api/data', (req, res) => {
 const requestBody = req.body;
 res.status(201).json({ message: 'Data received successfully', data:
requestBody });
});
```

This is an example of Middleware in an Express App. `(req, res, next) => {}` is a function that will perform actions on the request and response objects before the final handler is executed. The `next();` function passes control onto the middleware in the stack

```
app.use((req, res, next) => {
    console.log('Middleware executed');
    next();
});
```

Practical Uses in Web Projects

ExpressJS can be used in real-life instances such as

- Building RESTful APIs (aids in the creation of API endpoints and parsing through request data bodies, and returning JSON responses)
- Used in frameworks for single-page web applications (serves as backend and allows for frontend to consume API endpoints provided)

Live Demo Instructions (Workshop Component)

Part 1: Setup

- Create project folder

```
mkdir express-demo
cd express-demo
```

- Initialize the project

```
npm init
```

- Install Express

```
npm install express
```

- Make a public directory to hold your HTML and CSS files (the same files you used earlier in the semester)

```
    mkdir public
    cd public
    mkdir css
```

- While in `public`, create your `index.html` file and put your HTML code in there
- Then, in the `css` directory you've created, put your CSS code in there

Part 2: Writing the server

- Create a file called `server.js` in your project root
- Paste the following code in

```
// Load Express library
const express = require('express');
const path = require('path'); // We need to define a path so that Express
knows which static files to serve

// Create the app
const app = express();
const port = 3000;
```

```
// This is here for the static files
app.use(express.static(path.join(__dirname, 'public')));

// Start the server
app.listen(port, () => {
  console.log(`Server listening at http://localhost:${port}`);
});
```

Part 3: Run and Test

- Run server

```
node server.js
```

- Open http://localhost:3000

Boilerplate Code

```
// 1. IMPORT EXPRESS
const express = require('express');
const app = express();

// 2. MIDDLEWARE SETUP
// TODO: Add middleware to parse JSON requests
app.use(express._____()); // Hint: Use express.json()

// 3. ROUTES
// TODO: Create a GET route for '/'
app._____('/', (req, res) => {
  res.send(_____); // Hint: Send "Hello World"
});

// TODO: Create a POST route for '/submit'
app._____('/submit', (req, res) => {
  // Hint: Read data from req.body and send it back as JSON
  const userData = req._____;
  res._____(userData);
```

```
});

// 4. ERROR HANDLING (BONUS)
// TODO: Add 404 handler for invalid routes
app.use((req, res) => {
  res.status(_____).send('404 - Not Found');
});

// 5. START SERVER
const PORT = 3000;
app.listen(PORT, () => {
  console.log(`Server running on http://localhost:${_____}`);
});
```