**Module 1 Assignment 2: Basic Dynamic Web Project using Node.js and EJS - Creating and Displaying User Information**

Rafe Mohammed Abdul

Master’s in information technology, Arizona State University

IFT458&544: Middleware Prog & Database Sec

Prof: Dinesh Sthapit

August 27, 2023

1. **Research and Understand Node.js**

* Node.js is a free and open-source program that can run JavaScript code in a standalone application rather than a web browser. It allows for the development of extremely scalable and efficient online applications by utilizing the Google V8 JavaScript engine for server-side code execution. Its architecture is event-driven and non-blocking, allowing for asynchronous processing of requests. This improves both performance and responsiveness. For these reasons, as well as the fact that it can be used to build real-time apps and manage several connections at once, Node.js has become increasingly important in server-side programming.

1. **Installing Node.js**

A screenshot of a computer

Description automatically generated

1. **Explore Node.js Features**

Unlike browser-based JavaScript, Node.js provides additional functionality. It's great for constructing tools to manipulate files and server applications because it interacts with the local file system and allows for read/write operations. Additionally, Node.js can access databases directly, construct network servers, and make network queries, all of which are things that browser-based JavaScript cannot accomplish due to security concerns. However, unlike browser-side JavaScript, Node.js cannot access browser-specific APIs like the Document Object Model (DOM), restricting its ability to change web page content and respond to user interactions.

1. **Create a simple Node.js script.**

Input text file containing some lines.

A screenshot of a computer

Description automatically generated

Node.js script that rewrites the statements into uppercase and to a new output file.

A screenshot of a computer

Description automatically generated

1. **Reflection**

The ability to use JavaScript in a server-side context made Node.js an appealing platform on which to build and work. The learning curve for asynchronous programming was steep at first. In my opinion, Node.js is a crucial resource for developing streamlined, real-time online applications, as well as for managing data-intensive activities and bringing together frontend and backend developers.

**Req 4: Research HTTP Status numbers and provide a brief description of each of the status values.**

1. **1xx Informational:**

* 100 Continue: The server received the request headers and client can proceed with the request.
* 101 Switching Protocols: The server agrees to switch protocols as requested by the client.
* 102 Processing: A request has been received and is being processed by the server, but no answer has been received as of yet.
* 103 Early hints: Use this status code in conjunction with the Link header to allow the user agent to begin preloading resources as the server processes the request.

1. **2xx Successful:**

* 200 OK: The request was successful, and the server has returned the requested data.
* 201 Created: The request resulted in a new resource being successfully created.
* 202 Accepted: A response to the inquiry has not yet been made. It's indecisive because HTTP doesn't provide sending an asynchronous response to reveal the request's outcome. It is designed for batch processing or situations where a different process or server processes the request.
* 203 No Authoritative information: This status number indicates that the info returned was not obtained directly from the original server but rather from a local or third-party copy. Mirrors and backups of other resources are common applications for this. The 200 OK answer is recommended over this status except in that one circumstance.
* 204 No Content: The server successfully processed the request but there's no content to send in the response.
* 205 Reset Content: Requests the user agent to clear all changes made to the document that initiated the request.
* 206 partial Content: When a client sends a Range header to request a subset of a resource, this response code is returned.
* 207 Multi-Status: Information about numerous resources can be transmitted in cases when more than one status code is needed.
* 208 Already Reported: To prevent duplicating member information across several bindings to the same collection, this element can be used within a dav:propstat> response.
* 226 IM used: The server has processed a GET request for the resource and returned an instance-manipulation result for the current instance.

1. **3xx Redirection:**

* 301 Moved Permanently: The requested URL has been permanently moved to a new location.
* 302 Found (or 307 Temporary Redirect): The requested URL is temporarily located at a different address.
* 303 See other: This response was returned by the server to inform the client that they can access the requested resource by making a GET request to a different URI.
* 304 not Modified: For caching purposes, this is used. It assures the client that it can keep using the previously cached version of the response without worrying that it may have been updated.
* 305 use proxy: Originally defined in an older version of the HTTP protocol, this status code lets servers know that a proxy is required to retrieve the requested answer. Security issues with proxy settings in-band have led to its deprecation.
* 306 unused: This code for a reply is no longer in use and is being held for future use. Earlier versions of the HTTP/1.1 standard made use of it.
* 307 Temporary redirect: The server responds to direct the client to retrieve the resource at another URI using the same mechanism as before. This is the same as the 302 Found HTTP response code, except that the user agent must use the same HTTP method in both requests if a POST was used in the first.
* 308 permanent redirect: The Location: HTTP Response header indicates that the resource is permanently located at another URI. This is similar to the 301 Moved Permanently HTTP response code, but the user agent must use the same HTTP method in both requests if a POST was used in the first.

1. **4xx Client Errors:**

* 400 Bad Request: The server cannot understand the request due to a client error.
* 401 Unauthorized: Authentication is required and failed, or credentials are missing.
* 402 Payment Required:
* 403 Forbidden: The client does not have permission to access the requested resource.
* 404 Not Found: The requested resource could not be found on the server.

1. **5xx Server Errors:**

* 500 Internal Server Error: A generic error message indicating a server problem.
* 502 Bad Gateway: The server, while acting as a gateway or proxy, received an invalid response from the upstream server it accessed.
* 503 Service Unavailable: The server is not ready to handle the request due to temporary overloading or maintenance of the server.

These are just a few examples of HTTP status codes. They play a crucial role in communication between a client (e.g., web browser) and a server, indicating the outcome of a request and helping to diagnose and troubleshoot issues during web interactions.

**Req 5: Research HTTP Response headers and provide a brief description of each**

1. **Content-Type:**

* Description: Specifies the media type of the resource being sent in the response. Helps the client understand how to interpret the data.
* Example Value: Content-Type: application/json

1. **Content-Length:**

* Description: Indicates the length of the response body in bytes. Helps the client know how much data to expect.
* Example Value: Content-Length: 1024

1. **Location:**

* Description: Used in redirection responses (e.g., 3xx status codes) to indicate the new location to which the client should re-request.
* Example Value: Location: https://www.example.com/new-page

1. **Cache-Control:**

* Description: Controls caching behavior, specifying how and for how long the response can be cached by the client or intermediary caches.
* Example Value: Cache-Control: max-age=3600

1. **Date:**

* Description: Indicates the date and time at which the response was generated by the server.
* Example Value: Date: Thu, 26 Aug 2023 12:00:00 GMT

1. **Server:**

* Description: Specifies information about the server software or technology being used. Can provide insights into the server's configuration.
* Example Value: Server: Apache/2.4.29 (Unix)

1. **ETag:**

* Description: Provides a unique identifier (usually a hash) for a specific version of a resource. Helps with caching and conditional requests.
* Example Value: ETag: "123456789"

1. **Set-Cookie:**

* Description: Used to set cookies in the client's browser. Allows the server to send data that will be stored by the browser for future requests.
* Example Value: Set-Cookie: session\_id=abc123; Path=/; Secure; HttpOnly

1. **Access-Control-Allow-Origin:**

* Description: Used in Cross-Origin Resource Sharing (CORS) to specify which origins are permitted to access the resource.
* Example Value: Access-Control-Allow-Origin: https://www.example.com

1. **WWW-Authenticate:**

* Description: Sent by the server in a 401 Unauthorized response to indicate the authentication method that should be used by the client.
* Example Value: WWW-Authenticate: Basic realm="Example Realm"

HTTP response headers play a critical role in providing metadata about the response and controlling various aspects of communication between the server and the client. They ensure proper interpretation, handling, and security of the exchanged data.

**Running a Node.js application**

A screenshot of a computer

Description automatically generated

**TO Dos:**

**New Project**

A screenshot of a computer

Description automatically generated

**Code in the app.js**

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**Code in the index.ejs**

A screenshot of a computer screen

Description automatically generated

A screenshot of a computer

Description automatically generated

**Running the application**

A screenshot of a computer

Description automatically generated

**Final output**

A computer screen with a white screen

Description automatically generated

**Rubrics**

Id customerName phoneNumber address loanAmount interest loanTermYears loanType description Created 5 data elements based on the above specifications. Wrote the code in the index.ejs file to add the UI elements

A screenshot of a computer screen

Description automatically generated

A screenshot of a computer program

Description automatically generated

Created a computed field to calculate the total amount owed after the loanTermYears and display it in the UI.

A screenshot of a computer

Description automatically generated

A screenshot of a computer screen

Description automatically generated

Final Output:

A computer screen with a white background

Description automatically generated

A computer screen with a white background

Description automatically generated

**References**

<https://nodejs.org/en/about>

<https://developer.mozilla.org/en-US/docs/Web/HTTP/Status>

<https://learn.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2012-R2-and-2012/hh831707(v=ws.11)>

<https://www.youtube.com/watch?v=wJa5CTIFj7U>

<https://www.youtube.com/watch?v=iYM2zFP3Zn0>