**IFT 458/598: Middleware Prog & Database Security (Fall 2022)**

Saurabh Verma

Module 1 Assignment Introduction to Backend Web

Instructor: Dinesh Sthapit

September 03, 2022

**Steps performed while implementing Loan calculator:**

Step 1: Created 5 data elements in loanData.json file as shown below

Text

Description automatically generated

Step 2: Added UI elements to be shown on loan calculator web page:

A screenshot of a computer

Description automatically generated with medium confidence

Step 3: Used the below mentioned loan calculator formula,

Graphical user interface, text, application, Word

Description automatically generated

Code for calculating loan:

const httpServer = require('http');

const url = require('url');

const fs = require('fs');

const replaceTemplate = require('./modules/replaceTemplate');

/// Read data from file

// Template

const tempCourse = fs.readFileSync(

    `${\_\_dirname}/data/loanData.json`,

    'utf-8'

 );

 /////////////////////////////////

// Template

const templateHTMLCourse = fs.readFileSync(

    `${\_\_dirname}/template/templateCourse.html`,

    'utf-8'

  );

// //   function replaceTemplate(htmlStr, course){

// const replaceTemplate = (htmlStr, course)=>{ // fat arrow function or lambda

//     let output = htmlStr.replace(/{%NAME%}/g, course.courseName);

//     output = output.replace(/{%IMAGE%}/g, course.image);

//     output = output.replace(/{%FROM%}/g, course.from);

//     output = output.replace(/{%INSTRUCTOR%}/g, course.instructor);

//     output = output.replace(/{%CREDITS%}/g, course.credits);

//     output = output.replace(/{%DESCRIPTION%}/g, course.description);

//     output = output.replace(/{%ID%}/g, course.id);

//     return output;

// }

 const dataObj = JSON.parse(tempCourse);// string to JavaScript Object JSON

////////////////////////////////

//Create Server

// const server = httpServer.createServer(function (req, res) {// call back function

const server = httpServer.createServer( (req, res) =>{// call back function

    // const urlParameter = url.parse(req.url, true);

    // console.log(JSON.stringify(urlParameter.query));// convert to String

    // console.log(JSON.stringify(urlParameter.pathname));// convert to String

    const {query,pathname} = url.parse(req.url, true); // object distructors

    if(query.id){// if there is query parameter named id read as string

        // Courses page

        if (pathname === '/' || pathname.toLowerCase() === '/courses') {

            res.writeHead(200, {// Every thing ran successfully

                'Content-type': 'text/html'

            });

            const course = dataObj[Number(query.id)];// convert string to numeric value

            // const strCourseName = JSON.stringify(course);

           const totalLoanAmt = function (course){

                var Pmt = 250;

                 var monthInterest = course.interest/1200;

                 var Numbermonth = course.loanTermYears \* 12  ;

                var sample = (1+ monthInterest);

                var target = 1-(1/(Math.pow(sample,Numbermonth)));

                var LoanAmount = (Pmt/monthInterest)\*(target)

                return LoanAmount;

           }

            const finalAmt = totalLoanAmt(course);

        //console.log(course.loanAmount);

            let courseHTML = replaceTemplate(templateHTMLCourse, course);// function that will replace the course values in the HTML

            //   res.end(` We received our first request from the client at resource ${urlParameter.pathname.toLowerCase()} with query parameter ${urlParameter.query.id}

            //   ${JSON.stringify(course)}// convert object back to string

            //   `)

            //LoanAmt()

            courseHTML = courseHTML.replace(/{%LOANAMOUNT%}/g, finalAmt);

            res.end(courseHTML);

        }

    }

    else{

            res.writeHead(404, {// Server did not find what you were looking for

                'Content-type': 'text/html'

            });

            res.end(`resource not found`)

        }

    });

//Start Listening to requests

server.listen(8000, 'localhost', ()=> {

    console.log('Listening to requests on port 8000');

});

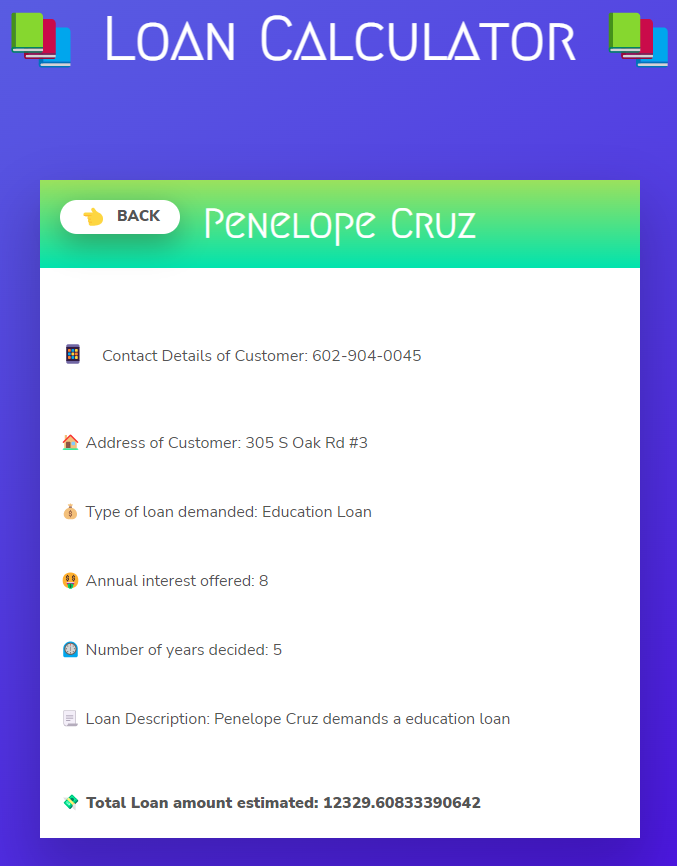
Screenshots of all the elements in the loanData.json file:

Graphical user interface, application

Description automatically generated

Graphical user interface, application

Description automatically generated



Graphical user interface, text, application

Description automatically generated

Graphical user interface, application

Description automatically generated

Step 4: Information about HTTP status numbers:

* Information responses:

100: This interim response instructs the client to proceed with the request or, if the request has already been completed, to disregard the response.

101: The protocol the server is moving to is indicated by this code, which is sent in response to an Upgrade request header from the client.

102: This code signifies that the request has been received and is being processed by the server, but no response has yet been made available.

103: To allow the user agent to begin preloading resources while the server is preparing a response, this status code is primarily designed to be used with the Link header.

* Successful responses:

200: The inquiry was granted. The HTTP method determines what "success" means as a result:

GET: The resource has been downloaded and sent in the body of the message.

HEAD: The response contains the representation headers but no message content.

PUT or POST: The message body transmits the resource that describes the action's outcome.

TRACE: The request message is contained in the message body as it was received by the server.

201: A new resource was produced because of the request's success. Usually, POST requests or some PUT requests receive this response.

202: The request was received, but nothing has been done about it yet. Since HTTP does not allow for the later sending of an asynchronous response revealing the outcome of the request, it is noncommittal. It is designed to be used for batch processing or situations where another server or process handles the request.

204: For this request, there is no content to provide, although the headers might be helpful. The user agent may substitute the new headers for those it has already cached for this resource.

Redirection messages:

300: There are multiple viable answers to the request. One of them should be selected by the user agent or user. (There isn't a set mechanism to select one of the answers; HTML links to the options are suggested instead so the user can decide.)

301: The requested resource's URL has been permanently modified. The answer contains the new URL.

304: This serves as a caching mechanism. It informs the client that the response hasn't changed, allowing the client to keep using the cached copy of the response.

Client error responses:

400: A perceived client fault causes the server to be unable to or unwilling to process the request (e.g., malformed request syntax, invalid request message framing, or deceptive request routing).

403: The server is refusing to provide the requested resource because the client does not have authorization to access the content. The client's identity is known to the server, unlike 401 Unauthorized.

429: The user sent out an excessive number of requests in a short period of time ("rate limiting").

451: A resource that cannot be legally delivered, like a website that has been banned by a government, was requested by the user agent.

Server error responses:

500: The server has come upon a circumstance that it is unsure how to handle.

502: This error message indicates that the server received an incorrect response while acting as a gateway to handle the request.

504: When the server is serving as a gateway and cannot get a response in time, an error response is returned.

505: The server does not support the HTTP version that was used in the request.

507: The server is unable to store the representation required to correctly finish the request, hence the method could not be applied to the resource.

511: Indicates that in order to access the network, the client must authenticate.

Step 5: Information about HTTP response headers:

UI Elements for HTTP Response Headers:

Feature page elements:

|  |  |
| --- | --- |
| **Name** | Shows the name component of a name-value pair in an HTTP response header. |
| **Value** | Shows the value component of a name-value pair in an HTTP response header. |
| **Entry Type** | Whether an item is local or inherited is indicated by the entry type. Both local and inherited items are read from the parent configuration file, whereas the current configuration file is used to read local items. |

Action Pane elements:

Add: opens the dialog box where you can add a header, called Add Custom HTTP Response Header.

Establish Common Headers: Sets the HTTP keep-alive and Web content expiration headers in the Set Common HTTP Response Headers dialog box.

Edit: opens a dialog box where you can update a header called Edit Custom HTTP Response Header. Only when an item has been chosen from the list on the feature page is this action accessible.

Remove: removes the selected item from the feature page's list.

Add or Edit Custom HTTP Response Header Dialog Box:

Name: In an HTTP response header name-value pair, type the name component.

Value: In an HTTP response header name-value pair, type the value component.

**References**

HTTP response status codes: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Status#information_responses>

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