Ryerson University Faculty of Engineering & Architectural Science

Department of Mechanical and Industrial Engineering

Program: Mechanical Engineering

Course Number	CCPS 530
Section Number	01
Course Title	Web Systems Development
Semester/Year	Fall 2021

Instructor	Dr. Ghassem Tofighi

Report NO.	1

Report Title	Final Exam

Group No.	N/A
Submission Date	December 12, 2021
Due Date	December 13, 2021

Name	Student ID	Signature*
Fareed Syed	xxxx19438	[illegible]

(Note: Remove the first 4 digits from your student ID)

GitHub Link to HTML page

https://github.com/FareedSyed31/FareedSyed31.github.io/blob/main/DictionaryWeb/index.js
https://github.com/FareedSyed31/FareedSyed31.github.io/tree/main/DictionaryWeb

HTML Tags and Usage

The following HTML tags were used in writing the HTML code for Final Exam.

1. **<!DOCTYPE>:** This was used to let the browser know what version of HTML doc is written.
2. **<html>:** This tag was used to set a container for other elements to be used in HTML documents.
3. **<head>:** This tag was used to define the head portion of the document containing info of HTML doc.
4. **<meta>:** This tag was used to set the info about the website for search engines.
5. **<body>:** This tag was used to define the main content of the HTML doc to display on the web browser.
6. <**style>:**This tag was used to set the font size and color of text on the HTML web page.
7. **<h1> :** This tag was used to set the name of the developer on the web page as Heading 1.
8. **<h2> :** This tag was used to set the profile and work experience on the web page as Heading 2.
9. **<img>:** This tag was used to set the image on the web page and edit the size of the image used.
10. **<p>:** This tag was used to create a paragraph of text in the body of the HTML code.
11. **<a>:** This tag was used to define the hyperlinks used to link from one page to another page.
12. **<b>:** This tag was used to bold the text within the tag range.
13. **<br>:** This tag was used to break the text to a new line within the paragraph of text.
14. **<ul>:** This tag was used to create an unordered list within the <h2> tag used for work experience.
15. **<li>:** This tag used to to create lists within the unordered list to specify details of work experience.
16. **<form>:** This tag is used to create an HTML form for user input.
17. **<label>:** This tag is used to create labels for several elements within the HTML.
18. **<input>:** This tag is used to create input elements for provided input types by HTML.
19. **<output>:** This tag is used to display the outcome of a user action performed in JavaScript.

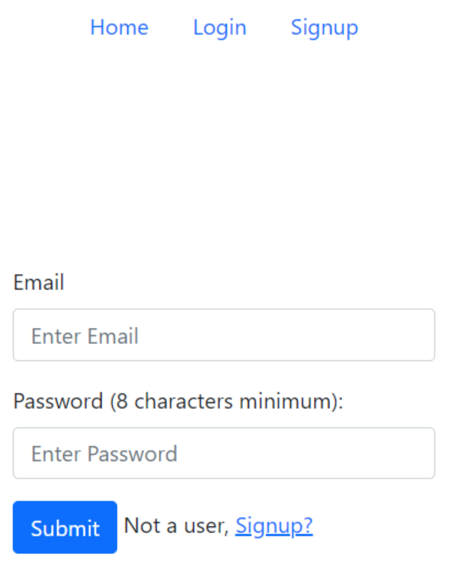
CSS Features

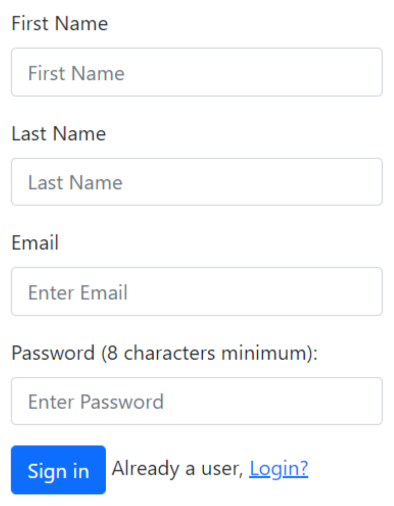
The following Bootstrap stylesheet was used for building this website.

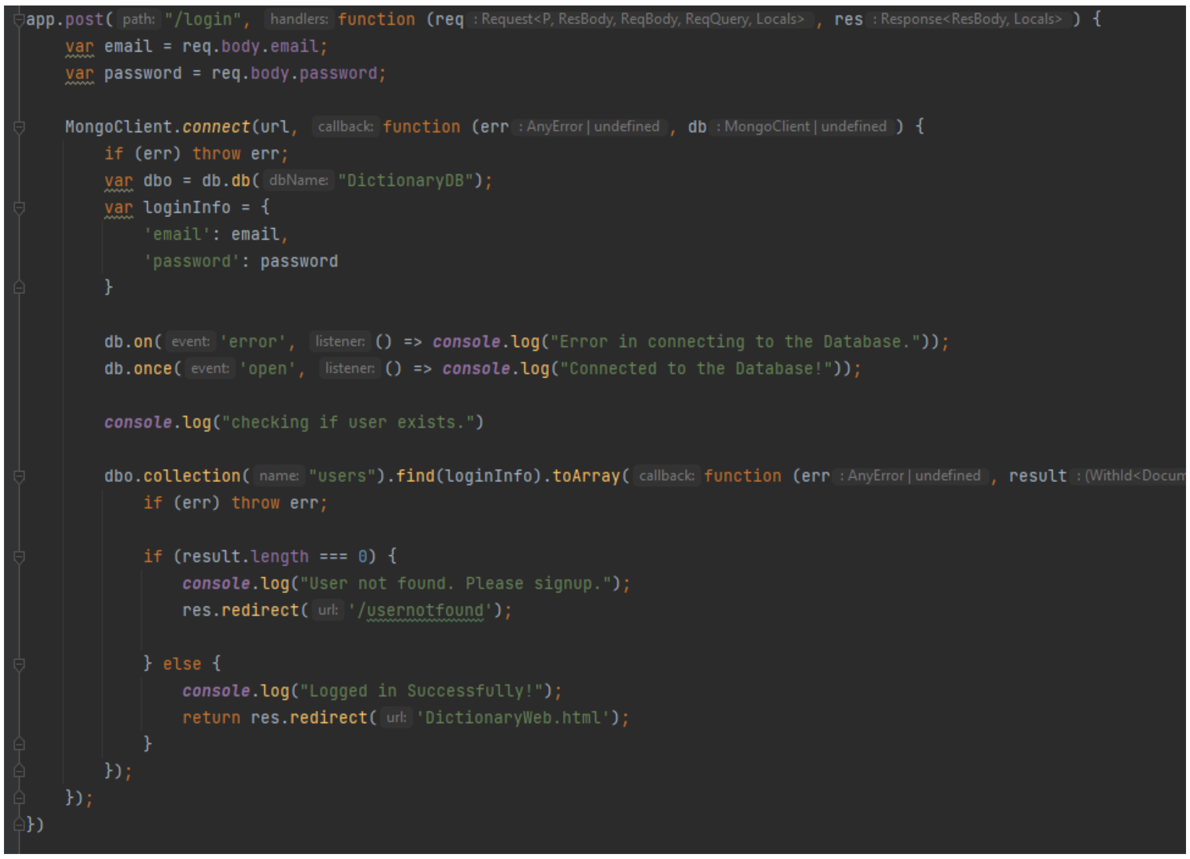
```
<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css"
rel="stylesheet"
     integrity="sha384-1BmE4kWBq78iYhFldvKuhfTAU6auU8tT94WrHftjDbrCEXSU1oBoqyl2QvZ6jIW3"
crossorigin="anonymous">

<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js"
       integrity="sha384-ka7Sk0Gln4gmtz2MlQnikT1wXgYsOg+OMhuP+IlRH9sENBO0LRn5q+8nbTov4+1p"
       crossorigin="anonymous">
```

User Login/Signup Route and Output UI

In order to access the website the user is required to make a new account and signup. If the user account exists then the user can simply login to the page.



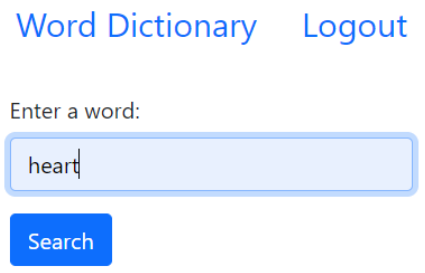




```
app.post("/login", function (req, res) {
    var email = req.body.email;
    var password = req.body.password;

    MongoClient.connect(url, function (err, db) {
        if (err) throw err;
        var dbo = db.db("DictionaryDB");
        var loginInfo = {
            'email': email,
            'password': password
        }

        db.on('error', () => console.log("Error in connecting to the Database."));
        db.once('open', () => console.log("Connected to the Database!"));

        console.log("checking if user exists.")

        dbo.collection("users").find(loginInfo).toArray(function (err, result
            if (err) throw err;

            if (result.length === 0) {
                console.log("User not found. Please signup.");
                res.redirect('/usernotfound');

            } else {
                console.log("Logged in Successfully!");
                return res.redirect('DictionaryWeb.html');
            }
        });
    });
})
```

Extracting JSON from Merriam-Webster Dictionary API

When a user enters a word in the provide text box for example in this case “heart”,

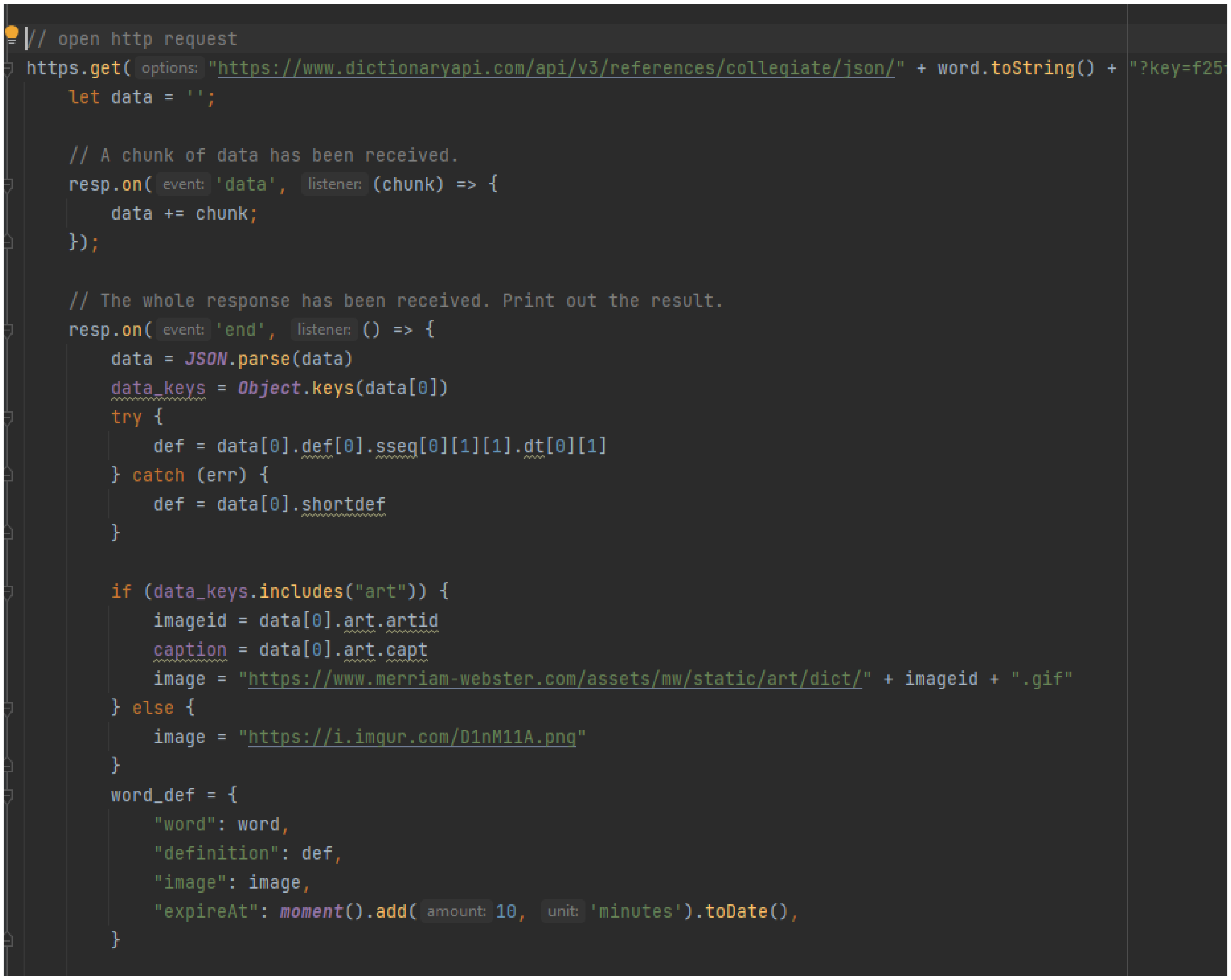

Word Dictionary Logout

Enter a word:

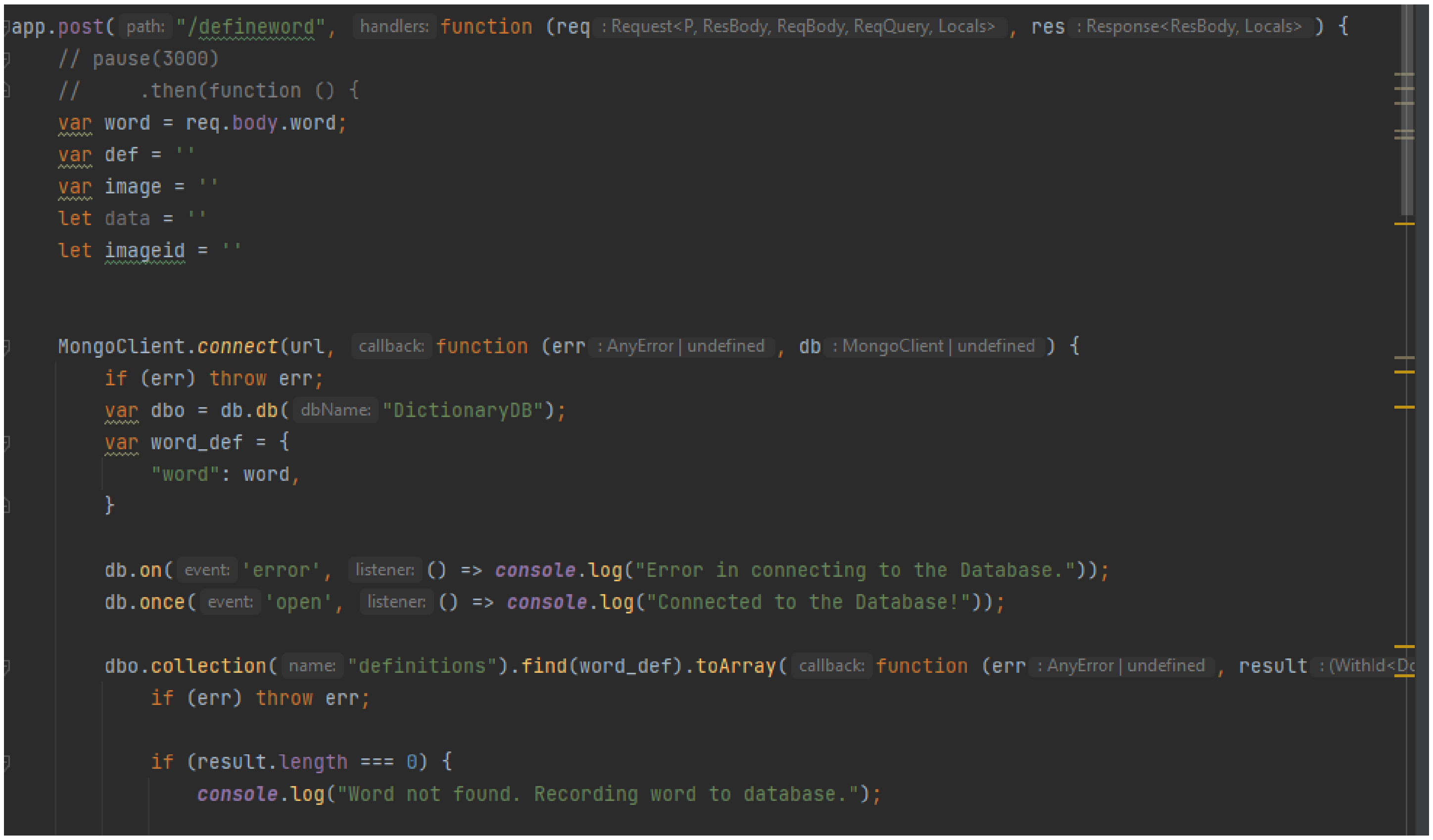
heart

Search

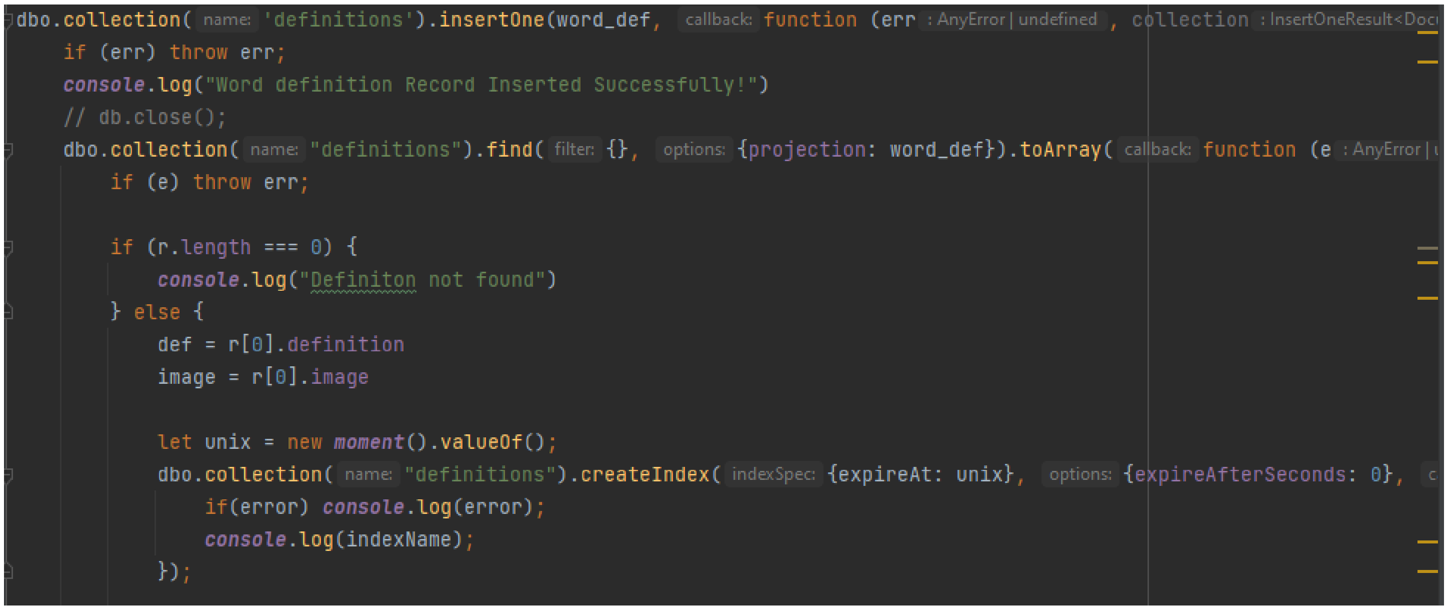
An HTTP request is opened in order to fetch JSON from the provided API. The following HTTP request is made.


```
// open http request
https.get( options: "https://www.dictionaryapi.com/api/v3/references/collegiate/json/" + word.toString() + "?key=f25
    let data = '';

    // A chunk of data has been received.
    resp.on( event: 'data', listener: (chunk) => {
        data += chunk;
    });

    // The whole response has been received. Print out the result.
    resp.on( event: 'end', listener: () => {
        data = JSON.parse(data)
        data_keys = Object.keys(data[0])
        try {
            def = data[0].def[0].sseq[0][1][1].dt[0][1]
        } catch (err) {
            def = data[0].shortdef
        }

        if (data_keys.includes("art")) {
            imageid = data[0].art.artid
            caption = data[0].art.capt
            image = "https://www.merriam-webster.com/assets/mw/static/art/dict/" + imageid + ".gif"
        } else {
            image = "https://i.imgur.com/D1nM11A.png"
        }
        word_def = {
            "word": word,
            "definition": def,
            "image": image,
            "expireAt": moment().add( amount: 10, unit: 'minutes').toDate(),
        }
```

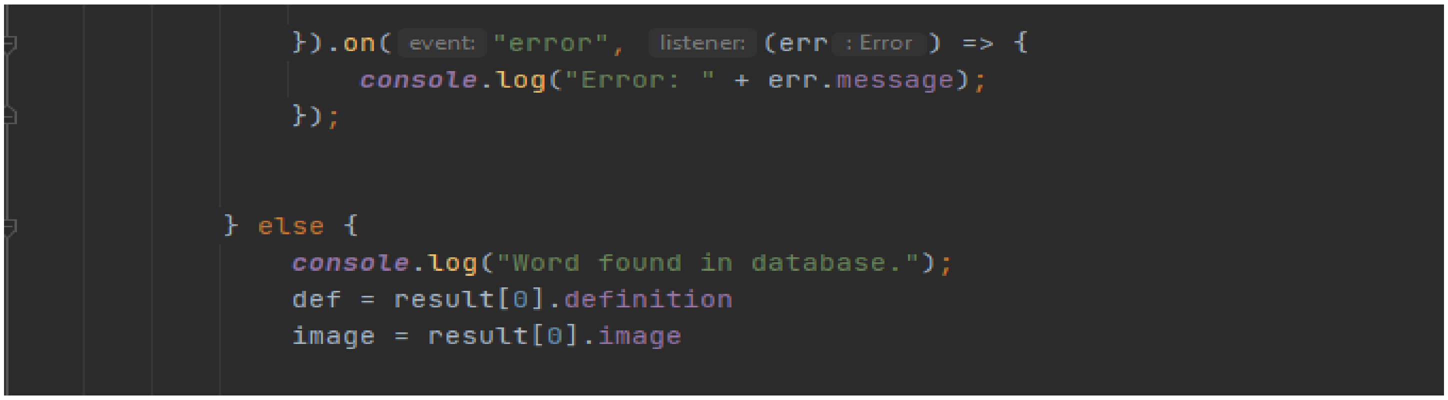
Once the data is received it is parsed into respective bits and such definition and illustrations of the object is parsed and the information is inserted as a record into the database with an expiry time of **10 min**. If the word was already searched by any user less than 10 minutes ago, and the same user or another user searches it again, just take the stored values from the database instead of the API call. If it does not exist in the database, take it from the API (to call the external API less and save costs).



```
app.post( path: "/defineword",  handlers: function (req : Request<P, ResBody, ReqBody, ReqQuery, Locals> , res : Response<ResBody, Locals> ) {
    // pause(3000)
    //     .then(function () {
    var word = req.body.word;
    var def = ''
    var image = ''
    let data = ''
    let imageid = ''


    MongoClient.connect(url,  callback: function (err : AnyError | undefined , db : MongoClient | undefined ) {
        if (err) throw err;
        var dbo = db.db( dbName: "DictionaryDB");
        var word_def = {
            "word": word,
        }

        db.on( event: 'error',  listener: () => console.log("Error in connecting to the Database."));
        db.once( event: 'open',  listener: () => console.log("Connected to the Database!"));

        dbo.collection( name: "definitions").find(word_def).toArray( callback: function (err : AnyError | undefined , result : (WithId<Do
            if (err) throw err;

            if (result.length === 0) {
                console.log("Word not found. Recording word to database.");
```

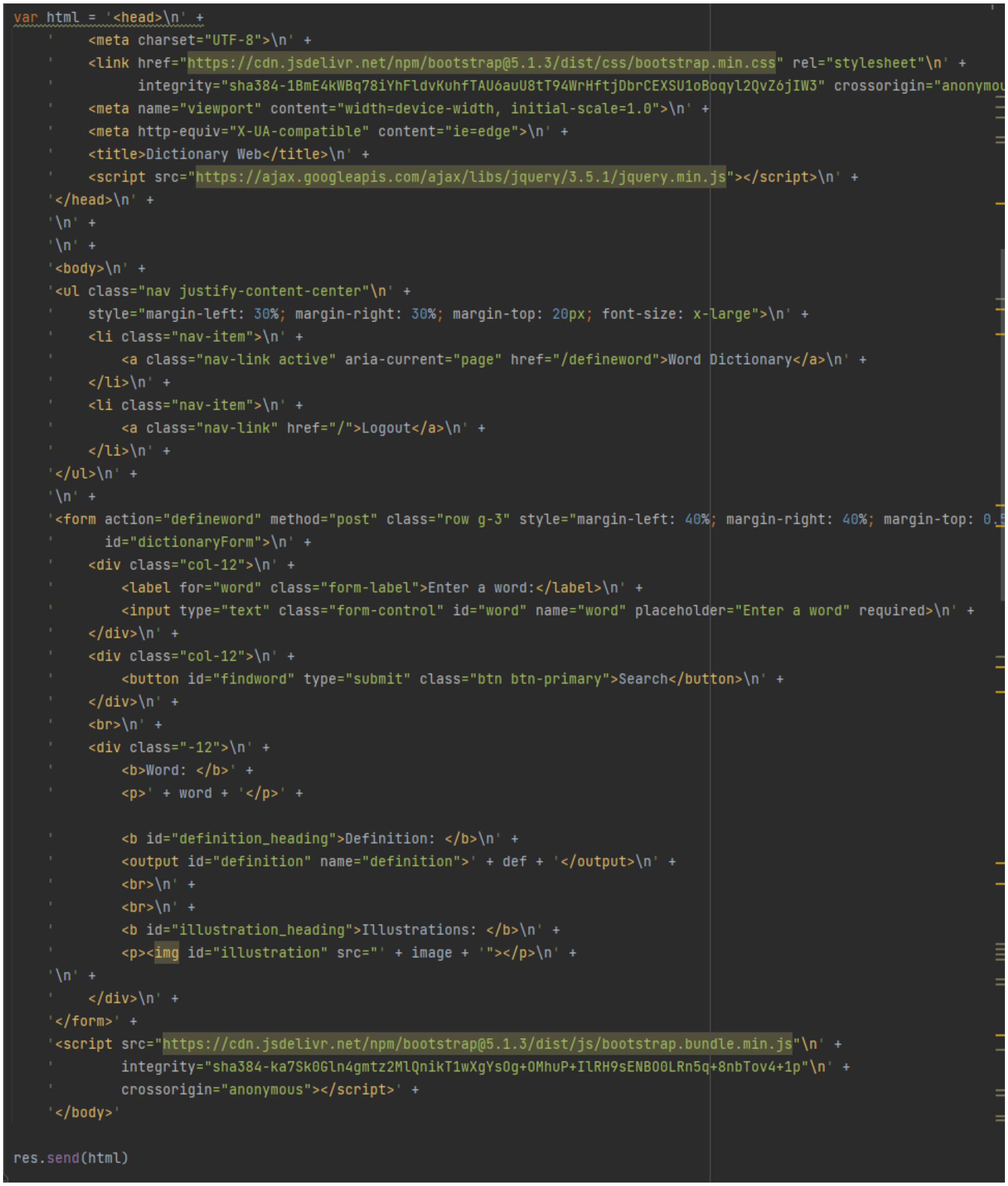


```
dbo.collection( name: 'definitions').insertOne(word_def,  callback: function (err : AnyError | undefined , collection : InsertOneResult<Doc
    if (err) throw err;
    console.log("Word definition Record Inserted Successfully!")
    // db.close();
    dbo.collection( name: "definitions").find( filter: {},  options: {projection: word_def}).toArray( callback: function (e : AnyError | 
        if (e) throw err;

        if (r.length === 0) {
            console.log("Definiton not found")
        } else {
            def = r[0].definition
            image = r[0].image

            let unix = new moment().valueOf();
            dbo.collection( name: "definitions").createIndex( indexSpec: {expireAt: unix},  options: {expireAfterSeconds: 0},  c
                if(error) console.log(error);
                console.log(indexName);
            });
```



```
            }).on( event: "error",  listener: (err : Error ) => {
                console.log("Error: " + err.message);
            });


        } else {
            console.log("Word found in database.");
            def = result[0].definition
            image = result[0].image
```

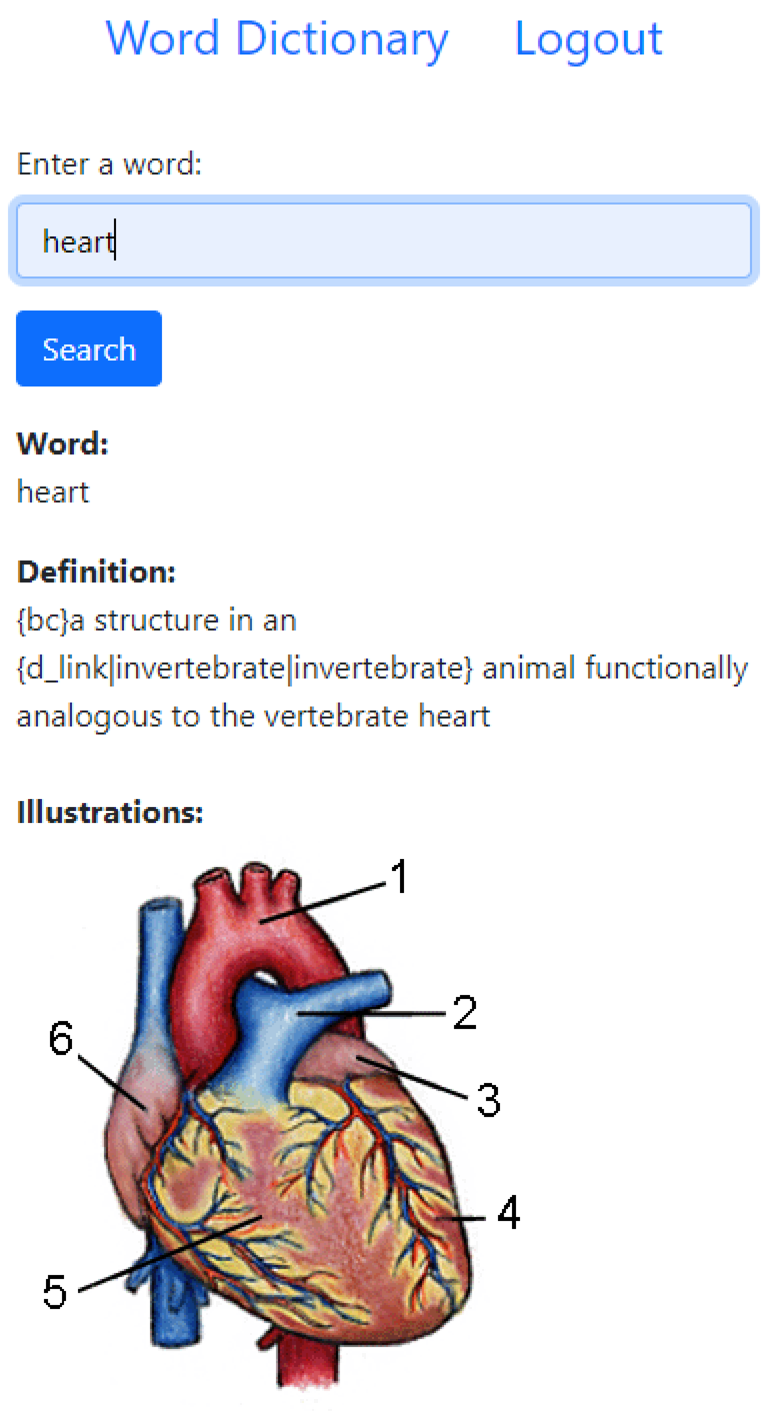
Once the information is safely recorded in the database, the definition and illustration of the word is provided to the **/defineword** route and with the help of **response.send()** the html is directed to the updated page. See below:



```
var html = '<head>\n' +
    '    <meta charset="UTF-8">\n' +
    '    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css" rel="stylesheet"\n' +
    '          integrity="sha384-1BmE4kWBq78iYhFldvKuhfTAU6auU8tT94WrHftjDbrCEXSU1oBoqyl2QvZ6jIW3" crossorigin="anonymou
    '    <meta name="viewport" content="width=device-width, initial-scale=1.0">\n' +
    '    <meta http-equiv="X-UA-compatible" content="ie=edge">\n' +
    '    <title>Dictionary Web</title>\n' +
    '    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>\n' +
    '</head>\n' +
    '\n' +
    '\n' +
    '<body>\n' +
    '<ul class="nav justify-content-center"\n' +
    '    style="margin-left: 30%; margin-right: 30%; margin-top: 20px; font-size: x-large">\n' +
    '    <li class="nav-item">\n' +
    '        <a class="nav-link active" aria-current="page" href="/defineword">Word Dictionary</a>\n' +
    '    </li>\n' +
    '    <li class="nav-item">\n' +
    '        <a class="nav-link" href="/">Logout</a>\n' +
    '    </li>\n' +
    '</ul>\n' +
    '\n' +
    '<form action="defineword" method="post" class="row g-3" style="margin-left: 40%; margin-right: 40%; margin-top: 0.5
    '      id="dictionaryForm">\n' +
    '    <div class="col-12">\n' +
    '        <label for="word" class="form-label">Enter a word:</label>\n' +
    '        <input type="text" class="form-control" id="word" name="word" placeholder="Enter a word" required>\n' +
    '    </div>\n' +
    '    <div class="col-12">\n' +
    '        <button id="findword" type="submit" class="btn btn-primary">Search</button>\n' +
    '    </div>\n' +
    '    <br>\n' +
    '    <div class="-12">\n' +
    '        <b>Word: </b>' +
    '        <p>' + word + '</p>' +

    '        <b id="definition_heading">Definition: </b>\n' +
    '        <output id="definition" name="definition">' + def + '</output>\n' +
    '        <br>\n' +
    '        <br>\n' +
    '        <b id="illustration_heading">Illustrations: </b>\n' +
    '        <p><img id="illustration" src="' + image + '"></p>\n' +
    '\n' +
    '    </div>\n' +
    '</form>' +
    '<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js"\n' +
    '        integrity="sha384-ka7Sk0Gln4gmtz2MlQnikT1wXgYsOg+OMhuP+IlRH9sENBO0LRn5q+8nbTov4+1p"\n' +
    '        crossorigin="anonymous"></script>' +
    '</body>'

res.send(html)
```

This is a very efficient process as this can be avoided by using .ejs or handlebars. As the developer was using IntelliJ IDEA as the IDE and it does not support .ejs and handlebars by default, it was a bit tough to render variables to the HTML. The developer also tried to download the respective plugins but was not able to work it out. ***Inefficient process but functional for the exam.***

Final Output



Enter a word:

heart

Word:

heart

Definition:

{bc}a structure in an {d_link|invertebrate|invertebrate} animal functionally analogous to the vertebrate heart

Illustrations:

Bootstrap vs CSS

This Final exam is a combination of styles used from CSS as well as Bootstrap. CSS and Bootstrap are free, front-end development frameworks designed to help developers build websites faster and easier. Major differences are that CSS3 is a less widely used framework that only uses CSS, while Bootstrap is a more widely used framework that uses CSS and JavaScript. CSS3 is considered an easier framework to learn and use for a few reasons. First, it is built only with HTML and CSS, which are easier to learn than other programming languages. Second, to load the CSS3 library on your site, you simply have to add a reference to the stylesheet in your index.html file.

With Bootstrap, you can choose not to use its JavaScript library and simply not load the JavaScript files on your site. The process of loading the CSS files is still more complicated than W3.CSS. Not only do you have to include a reference to the stylesheet, you also have to either load BootstrapCDN locally or download Bootstrap on your server. If you go with the latter, the download process is different depending on if you choose the pre-compiled or source code version.

Testing the RESTful APIs

REST API Testing is an open-source web automation testing technique that is used for testing RESTful APIs for web applications. The purpose of rest api testing is to record the response of rest api by sending various HTTP/S requests to check if rest api is working fine or not. Rest api testing is done by GET, POST, PUT and DELETE methods. However the common tests learnt so far are GET and POST methods in this course.GET– The GET method is used to extract information from the given server using a given URI. While using GET requests, it should only extract data and should have no other effect on the data. POST– A POST request is used to create a new entity. It can also be used to send data to the server, for example, customer information, file upload, etc. using HTML forms.

Consumption and Parsing of JSON

The JSON object extracted from the Merriam-webster dictionary api. This was done with the help of **JSON.parse(data)** function. The content from the object was loaded by parsing the information in the format of 'key':'value' pairs. The extracted info was placed successfully inside the MongoDB database and displayed on the webpage.

Web Browsers Used

The web browsers used during this Final Exam were Google Chrome and IntelliJ IDEA built in preview web browsers. When a web page is loaded, the browser first reads the HTML text and constructs a DOM Tree from it. Then it processes the CSS whether that is inline, embedded, or external CSS and constructs the CSSOM Tree from it. After these trees are constructed, then it constructs the Render-Tree from it. Different browsers use different rendering engines: Internet Explorer uses Trident, Firefox uses Gecko, Safari uses WebKit. Chrome and Opera (from version 15) use Blink, a fork of WebKit.

Conclusion

Total time spent on this Final Exam was around 20 hours. 2 hours were dedicated to readings, researching and learning using all the modules provided in the course, mainly Module 5 onwards, Stack Overflow and blog posts. And 10 hours were spent in writing the code. Another 4-5 hours were spent on debugging the code to make sure the HTML web page validated for HTML5 and CSS compliance. Approximately an hour was spent on writing the report for this Final Exam. Over all the Final Exam was a successful and fun learning experience. The video has been uploaded to youtube as an unlisted video and can be viewed here. Furthermore, it is also submitted in the submission folder.

References

A. Fitzgerald, "W3.CSS vs. Bootstrap: A Head-to-Head Comparison," *hubspot*, 02-Nov-2011. [Online]. Available: https://blog.hubspot.com/website/w3-css-vs-bootstrap. [Accessed: 23-Oct-2021].

Thomas Hamilton, "REST API Testing Tutorial: Sample Manual Test Cases & Rest API for Testing," *Guru99*, 8-Oct-2021. [Online]. Available: https://www.guru99.com/testing-rest-api-manually.html. [Accessed: 16-Nov-2021].

O. Emmanuel, "How browser rendering works - behind the scenes," *LogRocket Blog*, 30-Jul-2021. [Online]. Available: https://blog.logrocket.com/how-browser-rendering-works-behind-scenes/. [Accessed: 05-Oct-2021].