# 

# 

**Designing and developing a website**

**School: Sitrin**  
**Project Lead:** Haim Levhar  
**ID Number:** 329458632 **Year of Execution:** 2024

ACKNOWLEDGEMENTS

First and foremost, I would like to express my immense gratitude to the *Almighty* for granting me the opportunity, patience, and perseverance throughout the entire execution of the project, both during the learning phase and the implementation phase.

*Jon Duckett*, whose HTML and CSS book has significantly impacted this project, was invaluable to me during this time. I want to thank *Yishai Shobali*, a thoughtful person who helped me and suggested great ideas throughout this process. I would also like to thank *Ariel Levi,* a great person who helped me solidify my understanding of concepts along this journey. Thanks to *Eyal Menzel* for his suggestions of improvements and revisions that I did not notice, bringing this small side project to perfection. Additionally, I would like to acknowledge *Chat GPT* for its valuable assistance in handling the more tedious aspects of this process.

**Table of Contents**

[1](#_Toc184646886)

[Chapter 1 - Introduction to Web Development 3](#_Toc184646887)

[1.1 – Purpose and Overview 4](#_Toc184646888)

[1.2 - Key Steps in Website Design 4](#_Toc184646889)

[Chapter 2 - HTML Basics 5](#_Toc184646890)

[2.1 - Definition and Structure 6](#_Toc184646891)

[2.2 - Evolution of HTML 6](#_Toc184646892)

[2.3 - Core HTML Elements and Attributes 6](#_Toc184646893)

[Chapter 3 - CSS Fundamentals 7](#_Toc184646894)

[3.1 - Definition and Purpose 7](#_Toc184646895)

[3.2 - Integration with HTML 7](#_Toc184646896)

[3.3 - Selectors, Properties, and Declaration 7](#_Toc184646897)

[Chapter 4 - Web Development Components 8](#_Toc184646898)

[4.1 - Dynamic vs. Static Websites 8](#_Toc184646899)

[4.2 - Role of PHP and JavaScript 8](#_Toc184646900)

[4.3 - HTTP, HTTPS, and Web Browsers 8](#_Toc184646901)

[Chapter 5 - Design, Implementation and Source Code 9](#_Toc184646902)

[5.1 - Sitemap and Directory Structure 9](#_Toc184646903)

[5.2 - Example Code and Output Previews 12](#_Toc184646904)

[5.2.1 - Main Homepage 15](#_Toc184646905)

[5.2.3 - Bestsellers & New Arrivals 23](#_Toc184646906)

[5.2.4 - Contact 28](#_Toc184646907)

[5.2.5 - Visit 30](#_Toc184646908)

[5.3 – PHP & JSON: User Registration & Validation 38](#_Toc184646909)

[5.3.1 - Description of the Process 38](#_Toc184646910)

[HTML Elements Reference 42](#_Toc184646911)

[CSS Proprieties Reference 44](#_Toc184646912)

[Index 46](#_Toc184646914)

# 

# Chapter 1 - Introduction to Web Development

## 1.1 – Purpose and Overview

Create a website-proposal list of items whose values are different purposes for your website.

Refine and make a final decision on the item in the website-proposal list you would like to implement in your project. Examples include, but are not limited to: an online shop, news site, e-commerce site, and online services. In fact, this item will determine the purpose of your website, and therefore you must thoroughly examine the various items to choose the most appropriate and relevant one for you. Note that the previous statement applies only to building your own websites without the intention to launch them. In the real world, however, you must choose according to what you can offer.

After you have chosen the purpose of your website, you can then search on the Internet or any other resource regarding the information and services your website should provide.

## 1.2 - Key Steps in Website Design

Design a sitemap. Sitemaps are used to depict the structure of websites. This sitemap will be composed of the primary items on the main homepage's primary navigation menu and their child pages. It is recommended to duplicate the primary navigation menu of the website's main homepage for all of the pages in your website, whether they are child or descendant pages. This approach helps your visitors to return to the website's main homepage and other major pages with a simple click (for more details, refer to HTML&CSS Design and Build Websites: 461-462).

After you have completed designing a sitemap, you should diagram wireframes for all of the pages within the sitemap alongside a sitemap-based directory structure. The term

**wireframe** refers to a simple sketch of the key information that needs to go on each page of a site. This is a much more effective and efficient approach than making inconsistent architectures for similar web pages, slowing your progress and confusing your visitors. Key information can be the website's logo, navigation bar, headings, the main body of text, and user sign-in/up (for more details, refer to HTML&CSS Design and Build Websites: 463-464).

Add the HTML elements/CSS properties that cause webpages to look more attractive, thereby leveraging visitors to prefer to visit your site over others.

Beginthe process of implementing your project with HTML, CSS & JS code and give it existence!

# Chapter 2 - HTML Basics

## **2.1 - Definition and Structure**

**HTML** stands for **HyperText Markup Language**. A **hypertext** is any text that, when clicked, directs the user to the link specified. In other words, a hypertext moves the user from one **webpage** to another. **Markup languages** are used to annotate text with **elements**, treating each marked-up section according to the specified element. HTML is a simple, intuitive, and easy-to-use language aimed at defining the structure of webpages.

## **2.2 - Evolution of HTML**

Since the emergence of HTML 1.0, there have been five HTML versions. HTML 5.0 is the current version (as of 2024). HTML 1.0 was written by **Tim Berners-Lee** and first published in 1993. **HTML files should include a *<!DOCTYPE>* declaration, indicating web servers the HTML version used on the HTML document. For HTML5 files use *<!DOCTYPE html>*.**

## **2.3 - Core HTML Elements and Attributes**

Every HTML page must include three **tags**: <html>, <head>, and <body>. HTML documents consist of **objects**, which are the items or types of content found on a **webpage**, such as lists, text, forms, images, tables, links, videos, and audio files. **HTML attributes** provide additional instructions to modify their content and are inserted inside the **opening tags.**

# Chapter 3 - CSS Fundamentals

# 

## **3.1 - Definition and Purpose**

**CSS** stands for **Cascading Style Sheets**. CSS style files help control the appearance of webpages.

## **3.2 - Integration with HTML**

CSS code can be applied to HTML files by placing it within the HTML <style> tag or linking to separate **external CSS files** or using the *style* attribute inside HTML opening tags.

## **3.3 - Selectors, Properties, and Declaration**

**CSS selectors** are patterns or identifiers used to target specific HTML elements on a webpage to apply styles. They determine which elements the CSS rules enclosed within braces will affect. CSS supports various types of selectors, including:

* **Type selectors:** Target elements by their tag name (e.g., div, p).
* **Class selectors:** Target elements with a specific class attribute value, using a period (.) followed by the class name (e.g., .example).
* **ID selectors:** Target elements with a specific id attribute value, using a hash (#) followed by the ID name (e.g., #header).
* **Attribute selectors:** Target elements based on the presence or value of attributes (e.g., [type="text"]).
* **Pseudo-classes:** Target elements based on their state or position in the DOM (e.g., :hover, :first-child).

 **Declarations** in CSS specify how the element referred to by the selector should be styled. A declaration consists of a **property name** and a corresponding **property value**.

 Property names specify the element aspects you want to change, while property values define the setting for the chosen property.

# Chapter 4 - Web Development Components

## **4.1 - Dynamic vs. Static Websites**

**Static websites** consist of static HTML documents without functionality for uniquely identifying users or constructing personalized webpages. **Dynamic websites** offer these functionalities, providing more interactivity and customization for users.

## **4.2 - Role of PHP and JavaScript**

**PHP (Hypertext PreProcessor)** is a **server-side scripting language** used to generate dynamic HTML documents and it runs on the server machines. **JavaScript**, on the other hand, is a **client-side scripting language** used to generate dynamic content in response to user actions and it runs on the client machines. Adding JavaScript code to your site's repository makes your site more interactive and functional, improving the user experience and reducing server load.

## **4.3 - HTTP, HTTPS, and Web Browsers**

A **web browser** is software that allows users to access and display webpages on the **Internet**. The Internet can be seen as a network of networks, providing access to remote resources located on external networks. Launching a site requires a web server to store your files and send them when **HTTP (Hypertext Transfer Protocol) requests** come in. You can use your own server for this task or use the service of a **web hosting company**. Inputting a path to a website without specifying a file name results in the web browser directing the user to an HTML extension file named "index.html" (if it exists) within the specified directory.

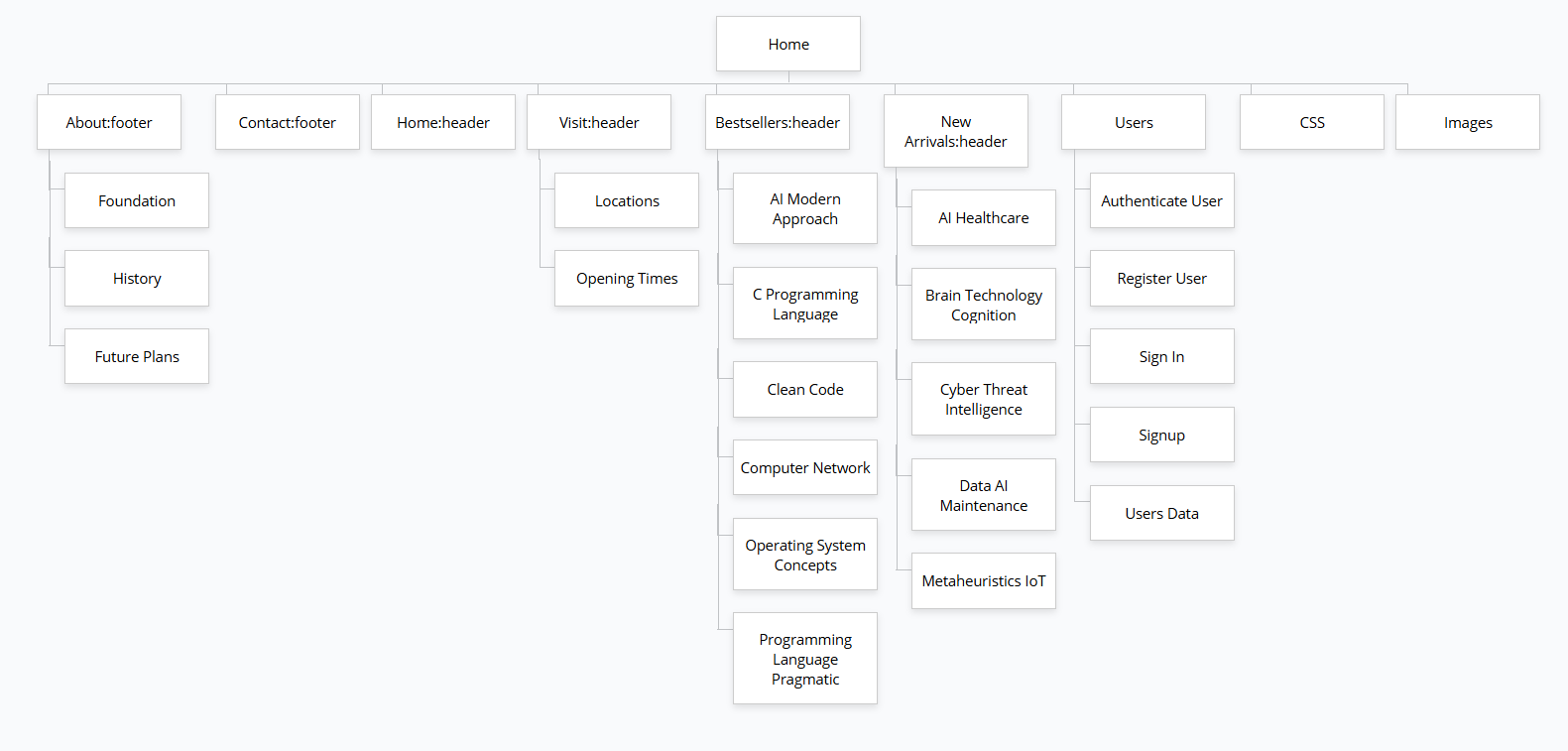
# Chapter 5 - Design, Implementation and Source Code

## 5.1 - Sitemap and Directory Structure

**Exploring, Listing, Refining, Deciding the Website's Purpose, Followed by Designing the Sitemap & Directory Structure.**

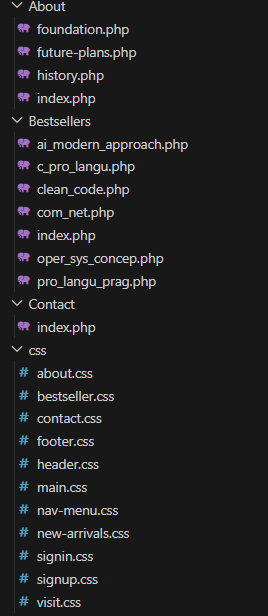
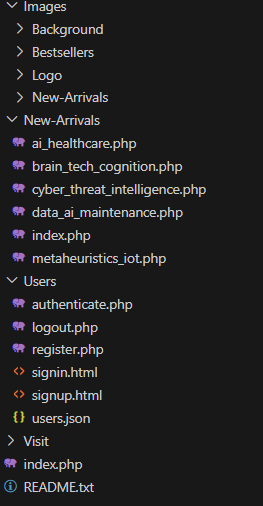
After thoroughly exploring the many possibilities for a website's purpose, I have decided that the website in this project will have the role and function of a simple Library Resource Portal. **A Library Resource Portal** is an online platform that provides access to a library's resources, such as catalogs, articles, maps, and other information, without offering books for sale. It helps users navigate and explore the library's physical and digital collections. One of the main reasons for this specific decision relates to the fact that designing and building a relatively small Library Resource Portal is neither too complex nor it is too simple but resides somewhere in between.

I have created necessary diagrams to help you see the big picture of the site, focusing on the **sitemap** and **directory structure**. The sitemap outlines the main sections of the site, such as the homepage, article pages, user register and authentication pages, and other types of files, along with their relationships. The directory structure diagram shows how these sections are organized within the site's files and folders, providing a clear view of how content is stored and accessed. These diagrams will help ensure the site’s layout is logical and well-organized and are located on the next few pages.

**Figure 5.1.1 – A Sitemap Depicting Key Pages for A Typical Library Resource Portal**

|  |  |
| --- | --- |
| **Section Hierarchy Level** | **Related Pages** |
| **Root Directory** | Homepage |
| **Child Directories** | About, Contact, Home, Visit, Bestsellers, New Arrivals, Users, CSS, Images |
| **HTML, CSS, PHP, & JSON Files.** | History, Foundation, Future Plans, Locations, Opening Times, AI Modern Approach, C Programming Language, Clean Code, Computer Networks, Operating System Concepts, AI Healthcare, Brain Technology Cognition, Cyber Threat Intelligence, Data AI Maintenance, Metaheuristics IoT, Authenticate User, Register User, Sign in, Signup, Users Data, All CSS files, All Images. |

**Figure 5.1.2 – Website Section hierarchy and navigation**

**MentoraNexus Folder:**

**Figure 5.1.3 – MentoraNexus website's directory structure**

## 

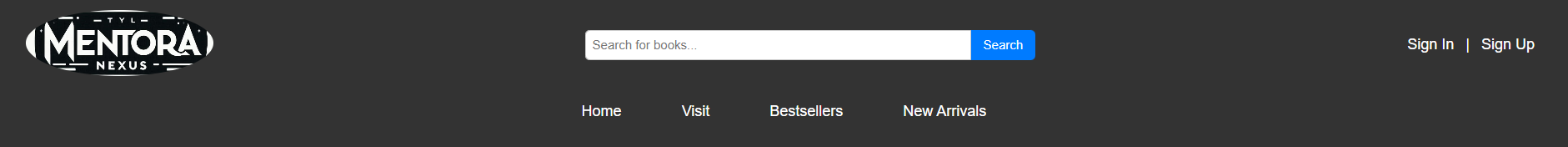
## 5.2 - Example Code and Output Previews

This section provides a structured presentation of the **source code** and corresponding visual output for key components of the Library Resource Portal. Each subsection is dedicated to a specific functionality or folder within the root directory, including the homepage, navigation menus, user registration, and dynamic content rendering.

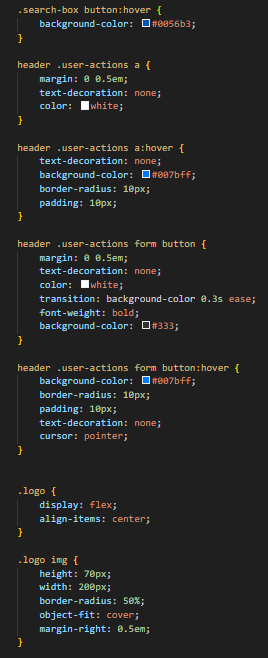
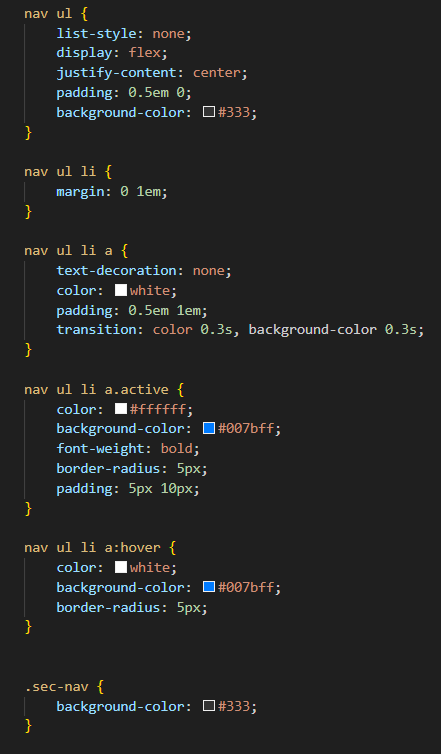
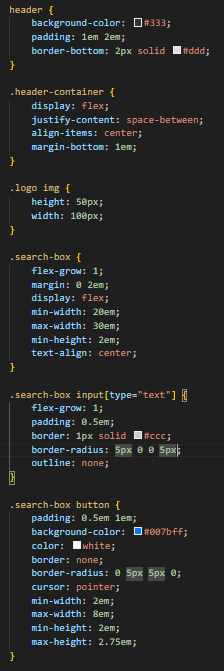
For example, the code for the navigation menu, located in the header.html and styled with nav-menu.css, demonstrates how to create a responsive and visually appealing interface. The output preview shows the navigation menu seamlessly integrating links to essential pages, such as "Visit", "Bestsellers", and "New Arrivals," ensuring consistent user experience across the portal.

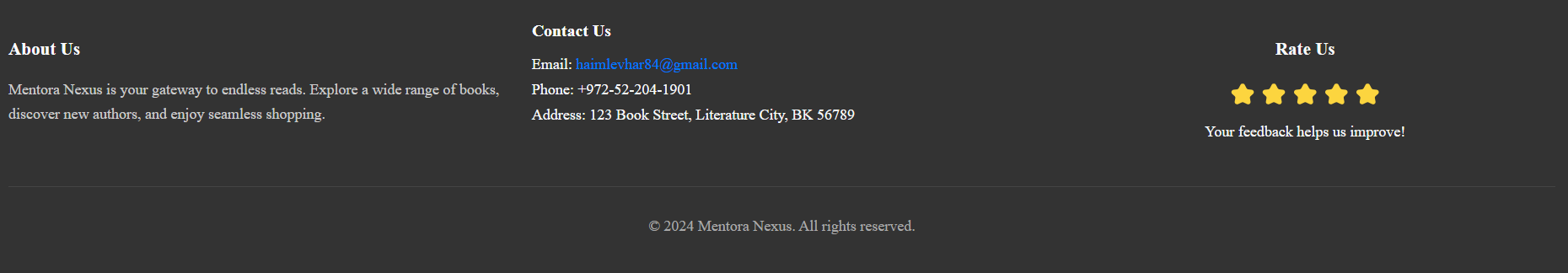
Similarly, the user registration and authentication functionality, implemented in signup.html and signin.html and styled with signup.css and signin.css, illustrates the use of **form elements** and **validation scripts** to enhance user interaction. Output previews depict the intuitive layout and responsiveness of these forms, emphasizing accessibility and ease of use.

These examples not only highlight the implementation of core features but also serve as practical guides for translating theoretical concepts into functional and aesthetically coherent website components. Readers are encouraged to analyze the provided source code and adapt the demonstrated techniques to their own web development projects

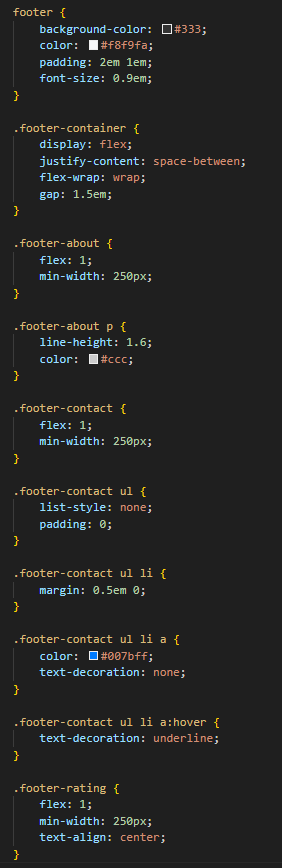
****Source Code 5.2.1 – header-html:

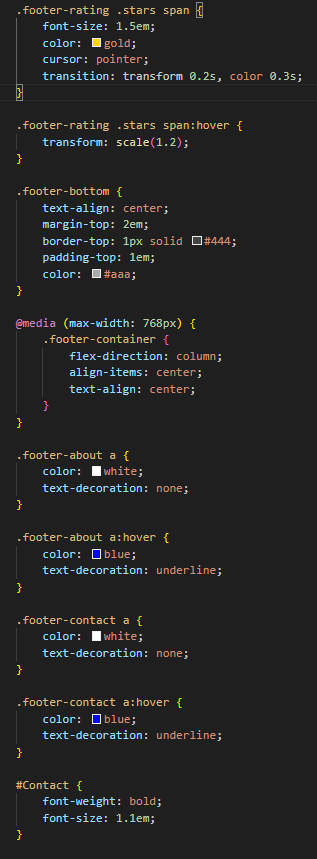
****

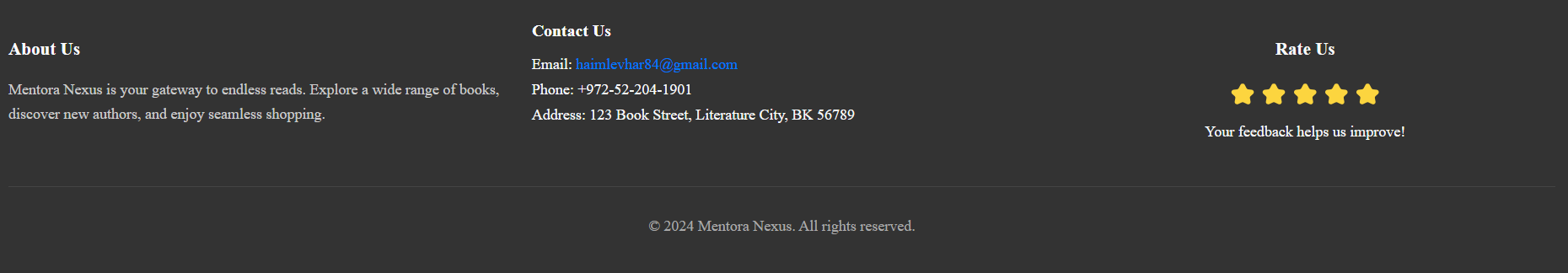
Source Code ****5.2.2 - header.css & nav-menu.css:

Source Code ****5.2.3 – footer-html:

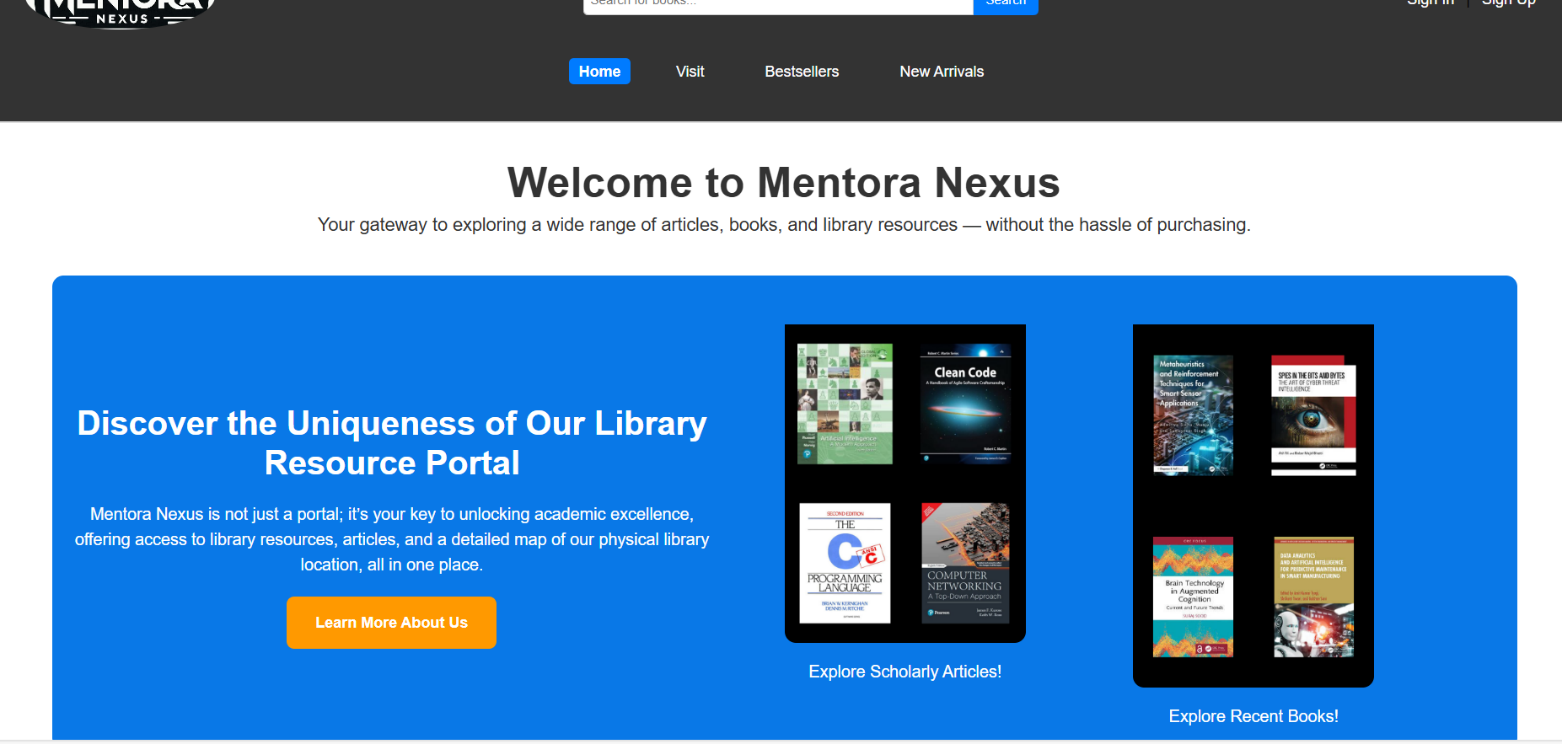


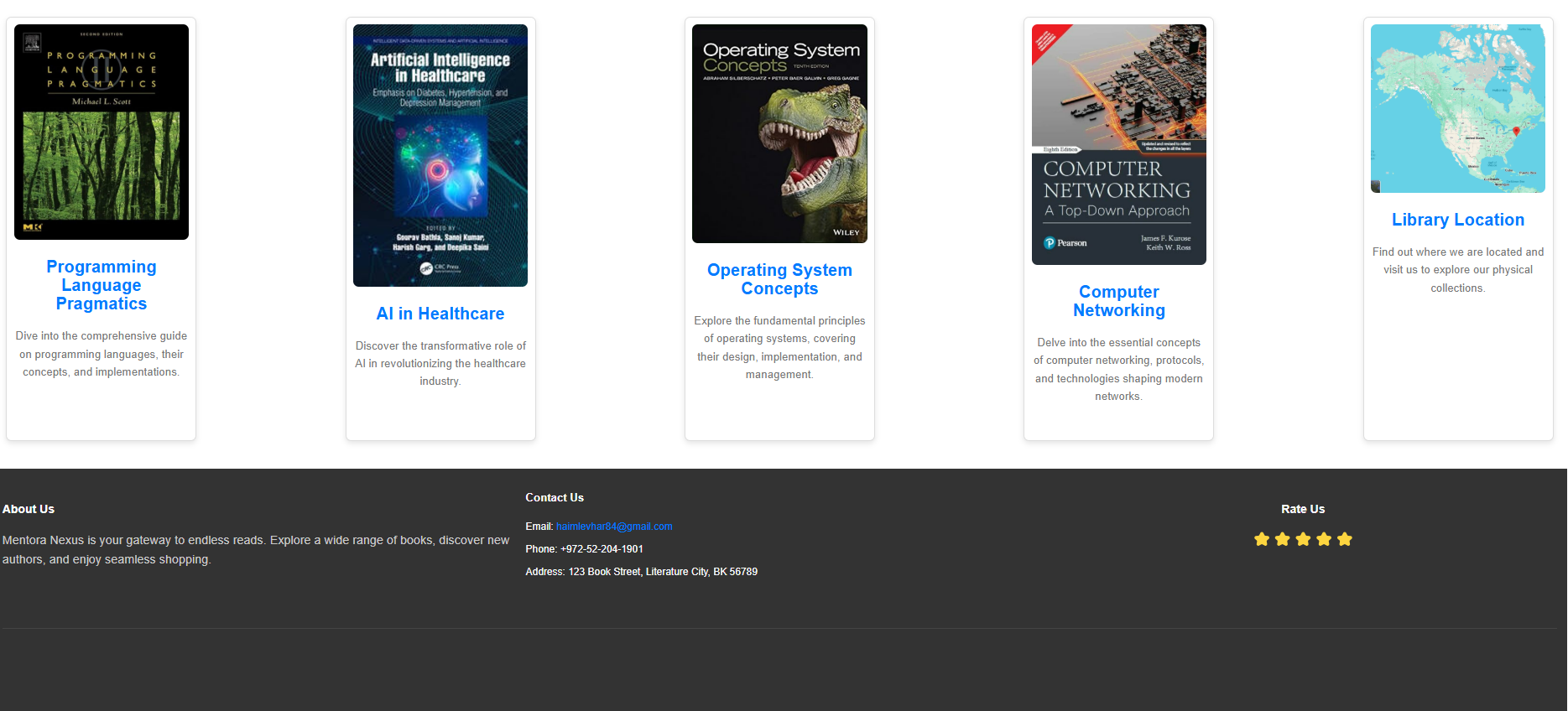
Source Code 5.2.4 - footer.css



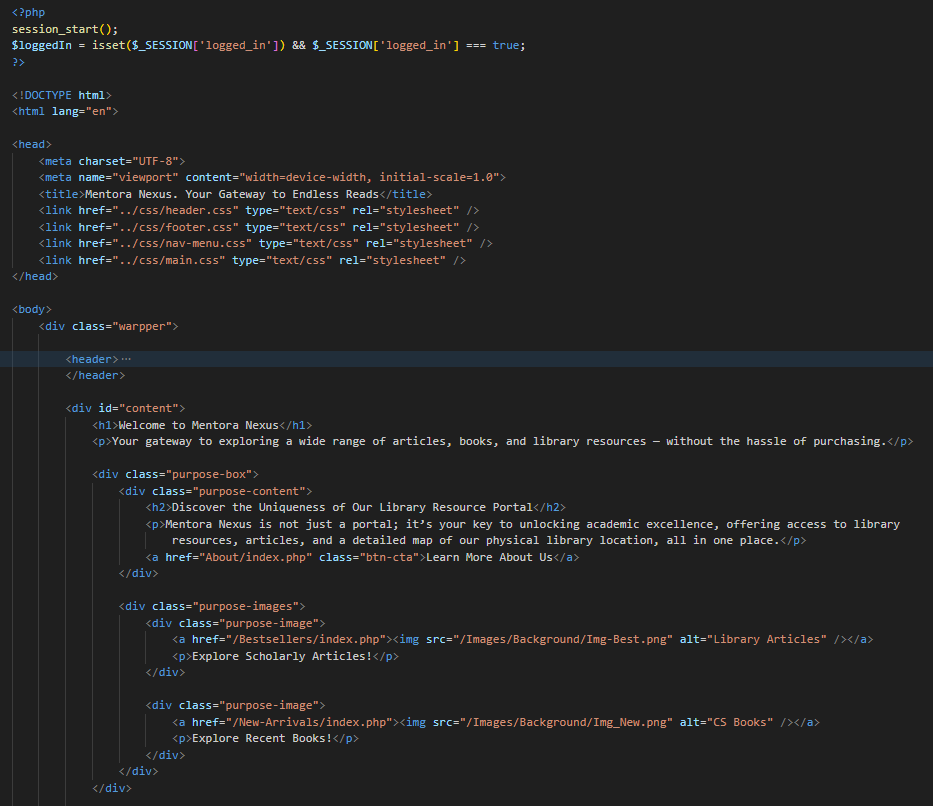
****

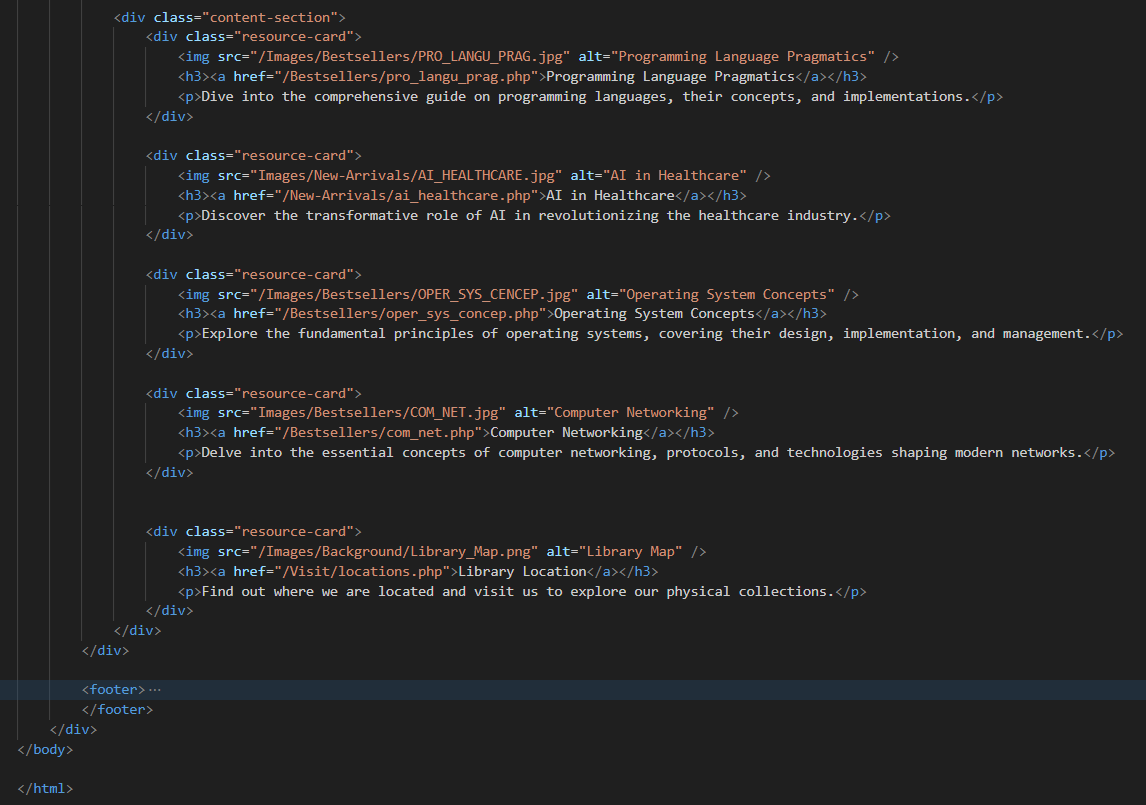
### 5.2.1 - Main Homepage

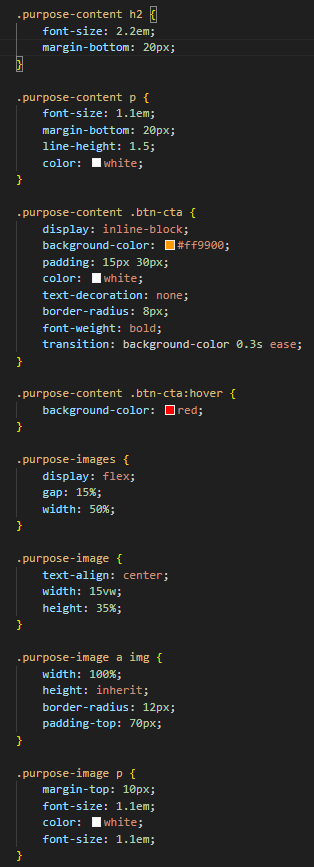
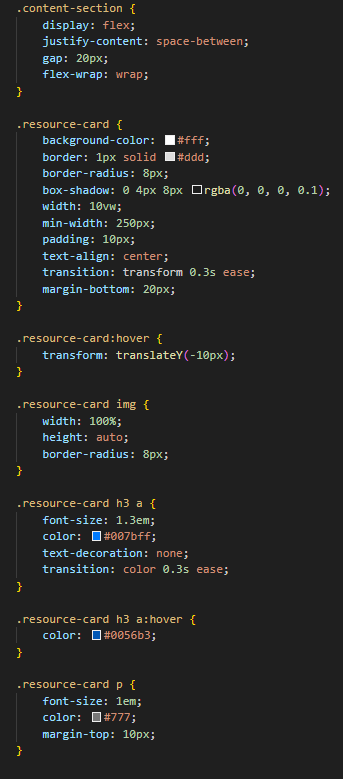
**Figure 5.2.1.1 – Main Homepage of MentoraNexus **

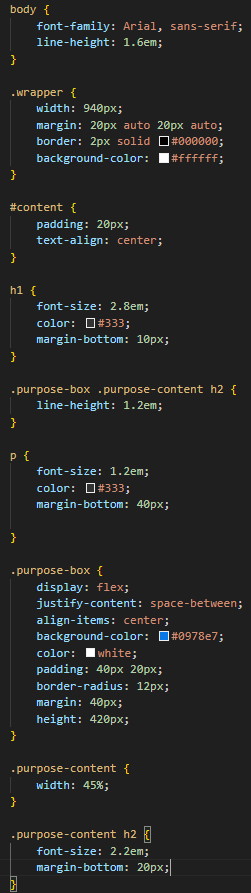
****

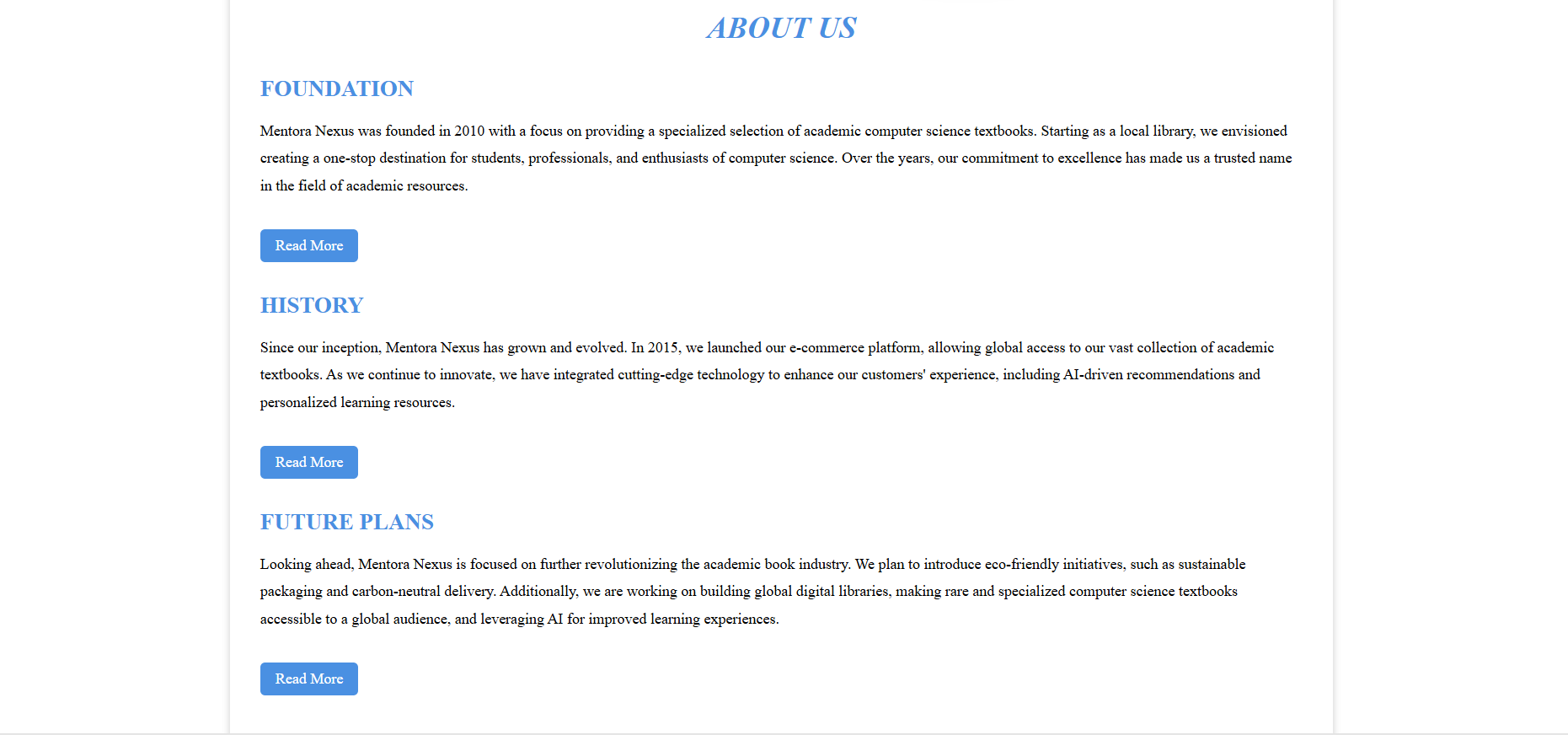
Source Code 5.2.1.1 – index.php:

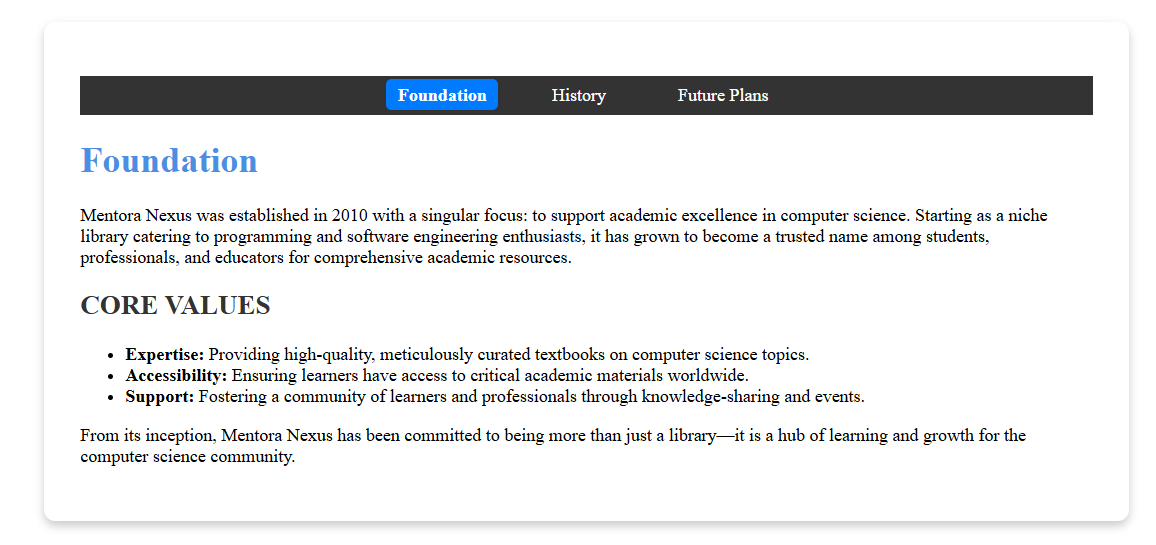


****

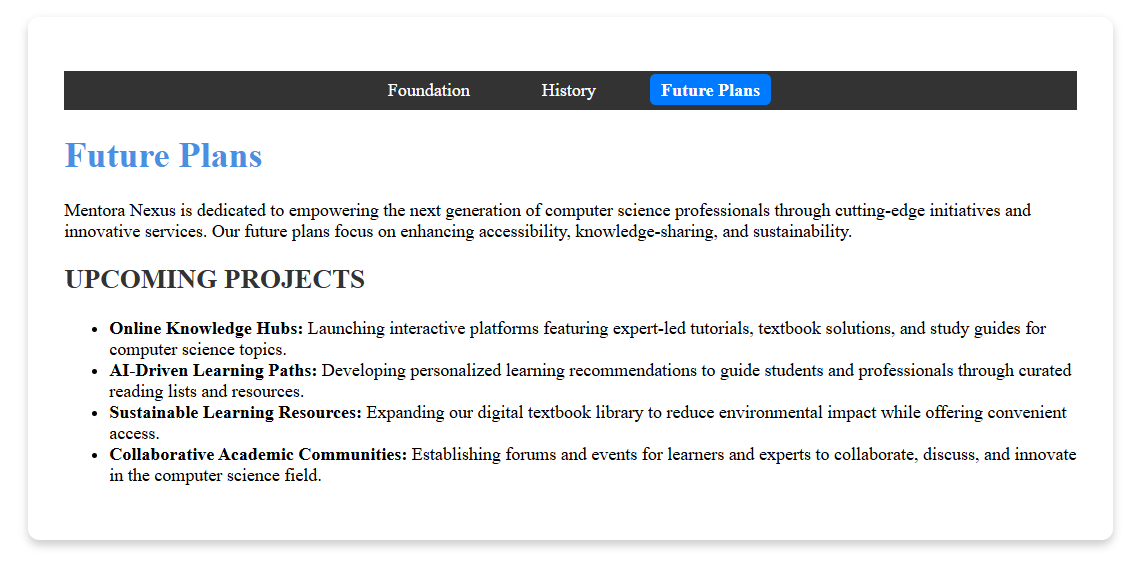
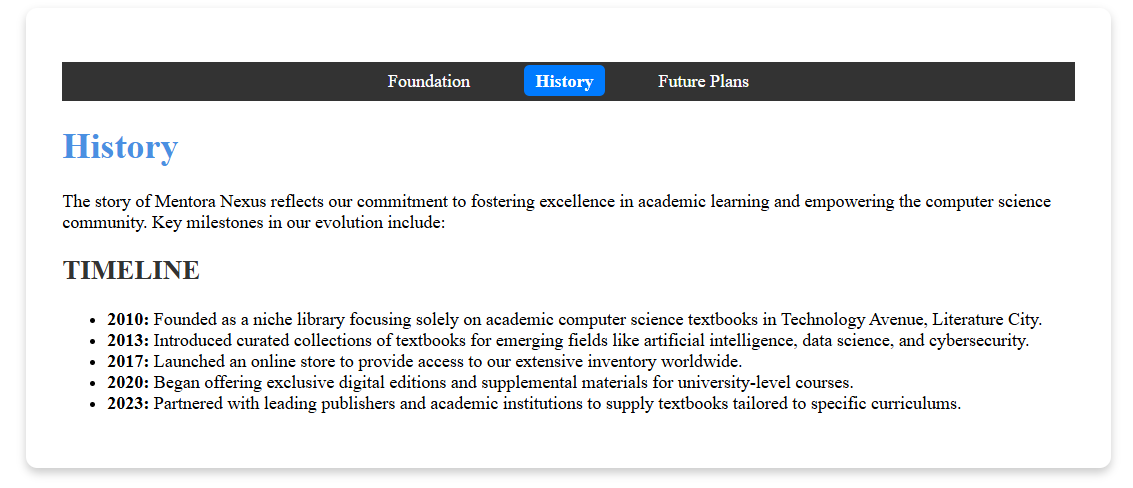
Source Code ****5.2.1.2 – main.css:

****

**** 5.2.2 - About

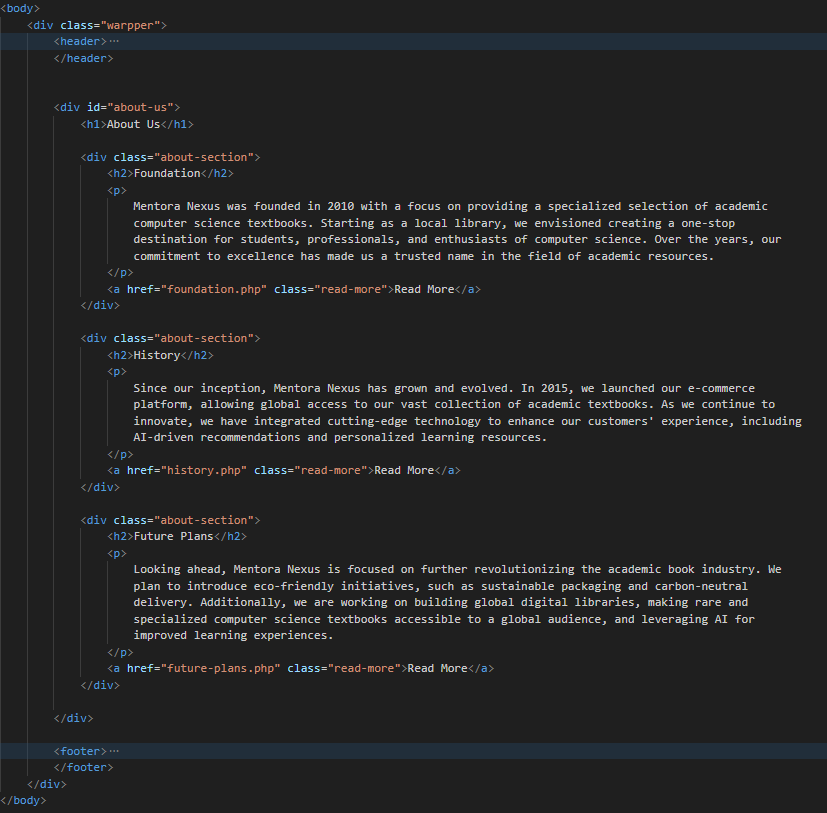
**Figure 5.2.2.1 – About Homepage**

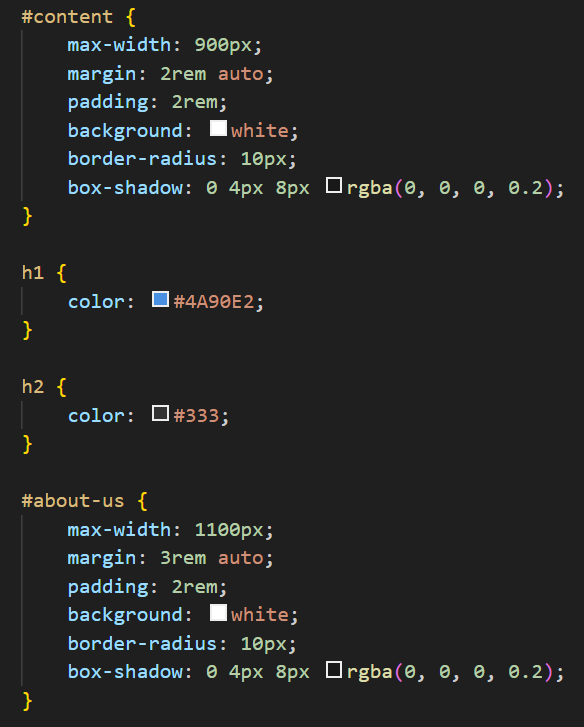
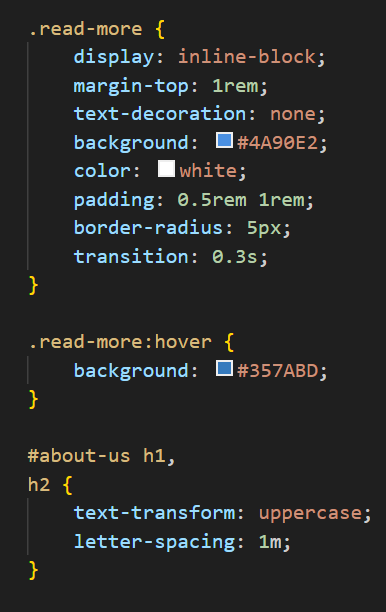
**Figure 5.2.2.2 – Foundation Section**

**Figure 5.2.2.3 – History Section**

**Figure 5.2.2.4 – Future Plans Section**

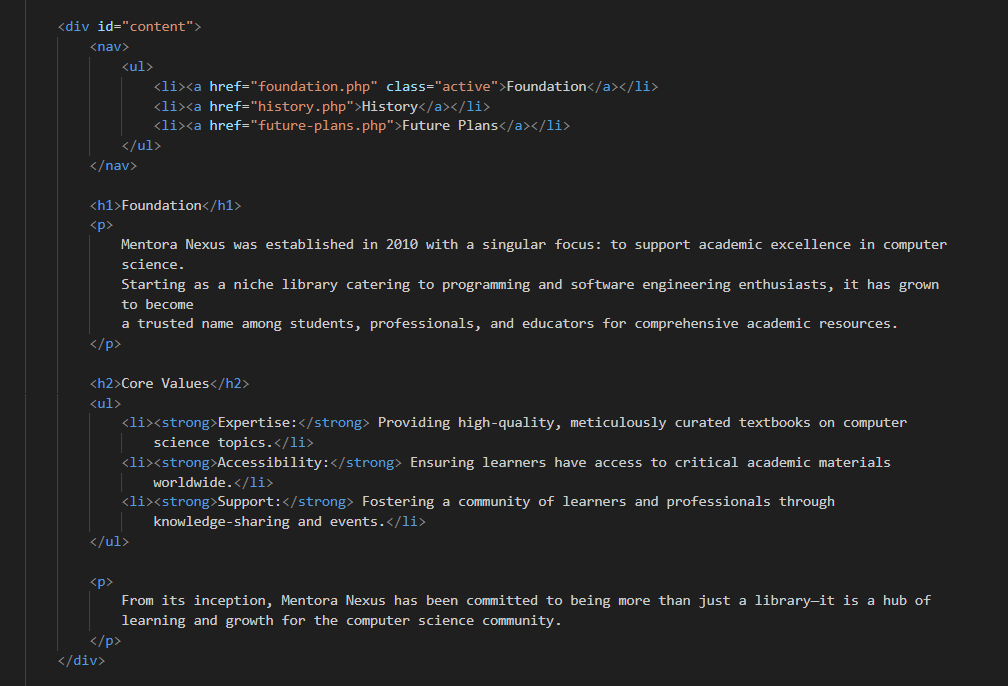
Source Code 5.2.2.1 – index.php:



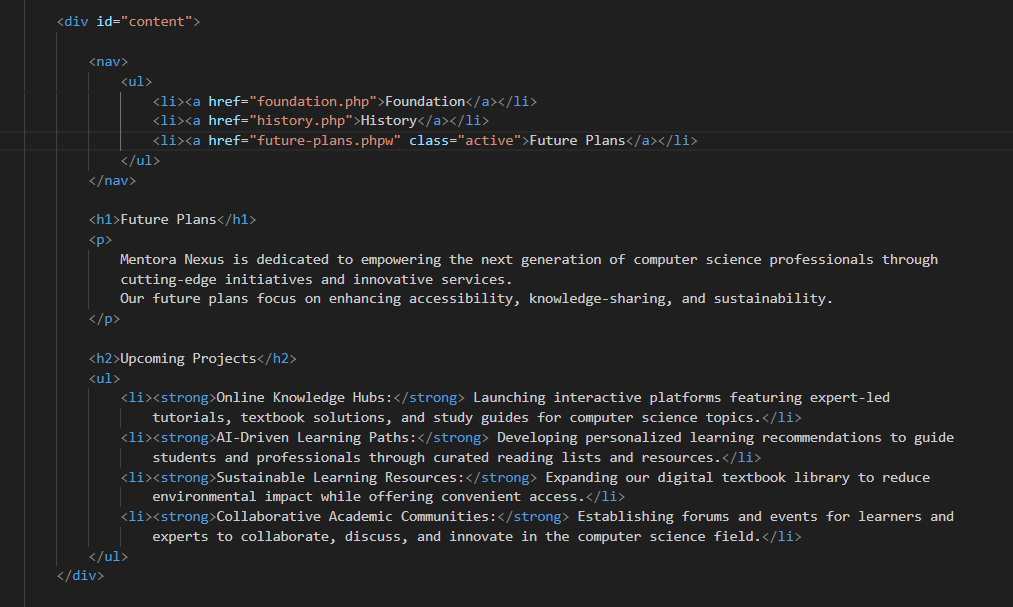
****Source Code 5.2.2.2 – about.css:

****

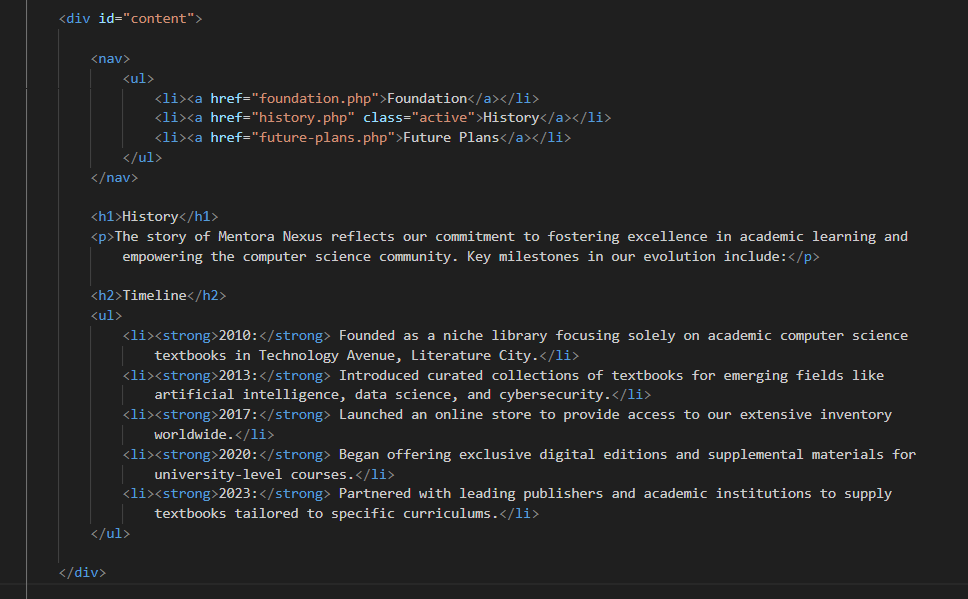
Source Code 5.2.2.3 – foundation.php:

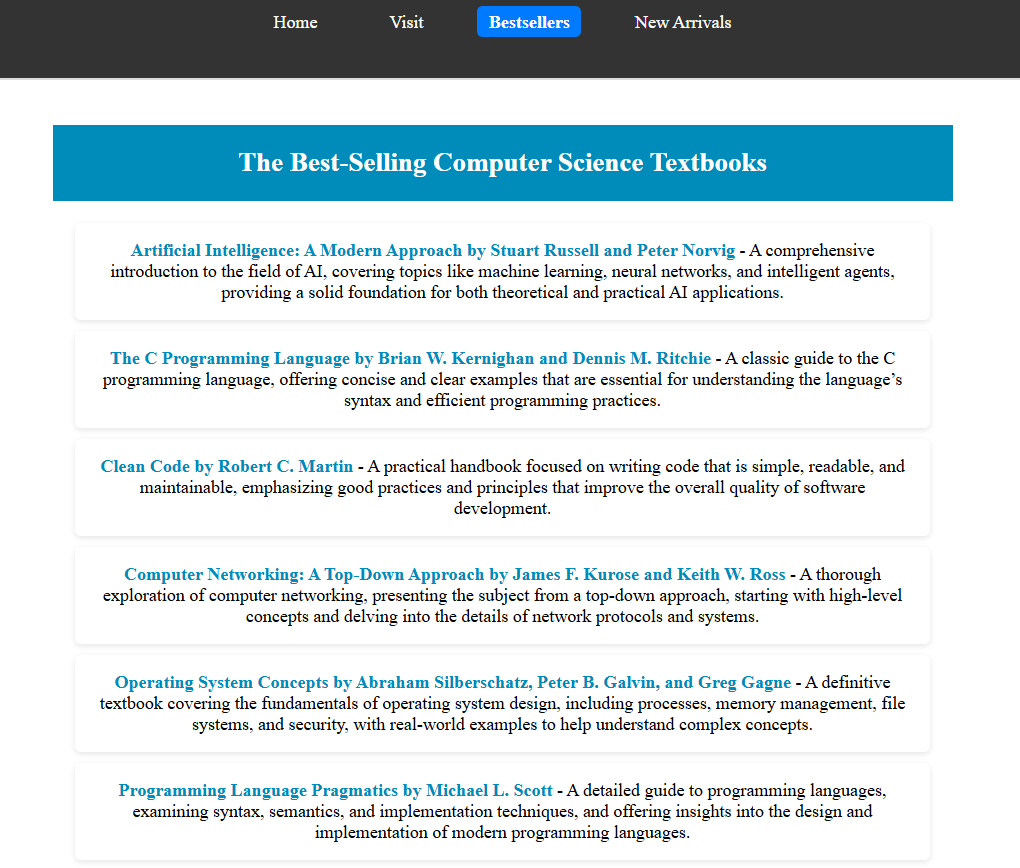
****

Source Code 5.2.2.4 - history.php:

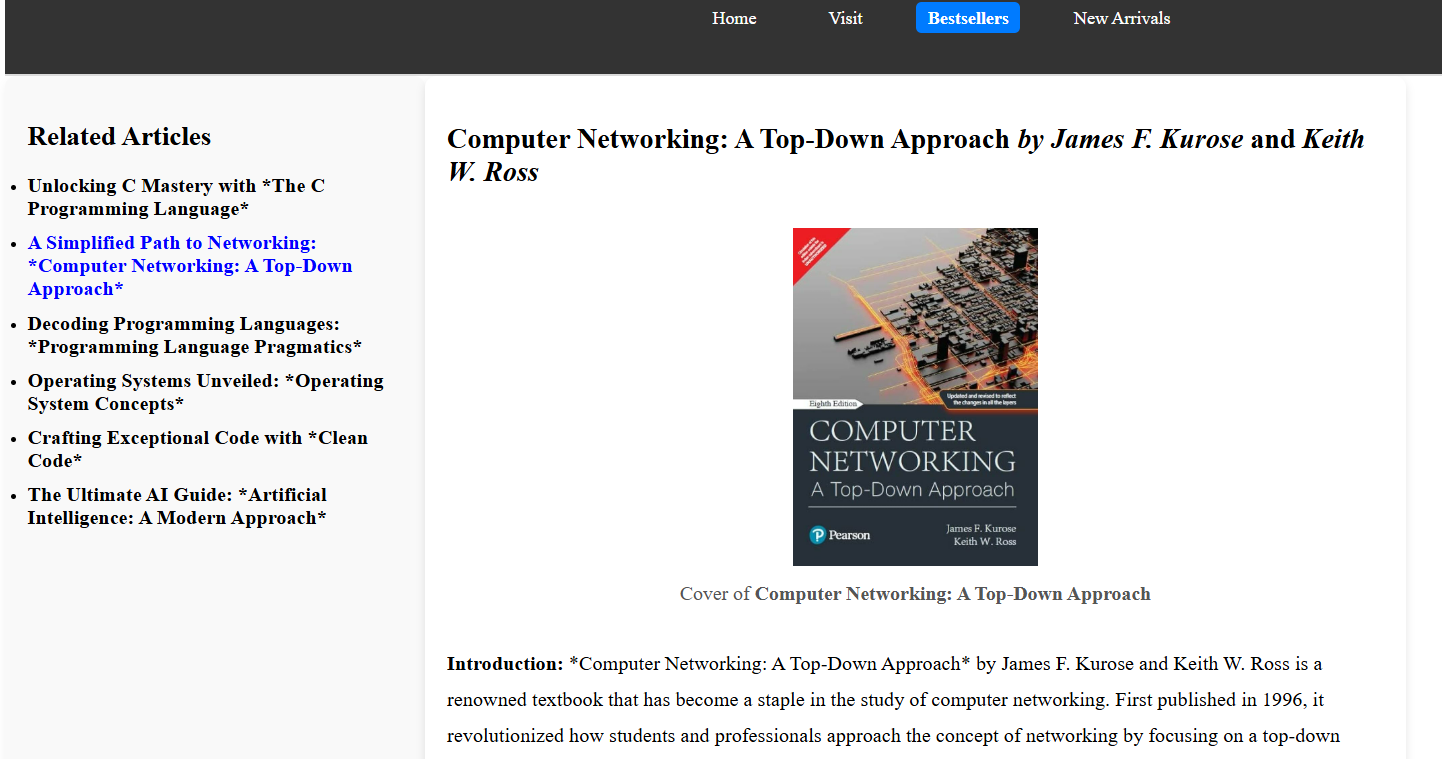
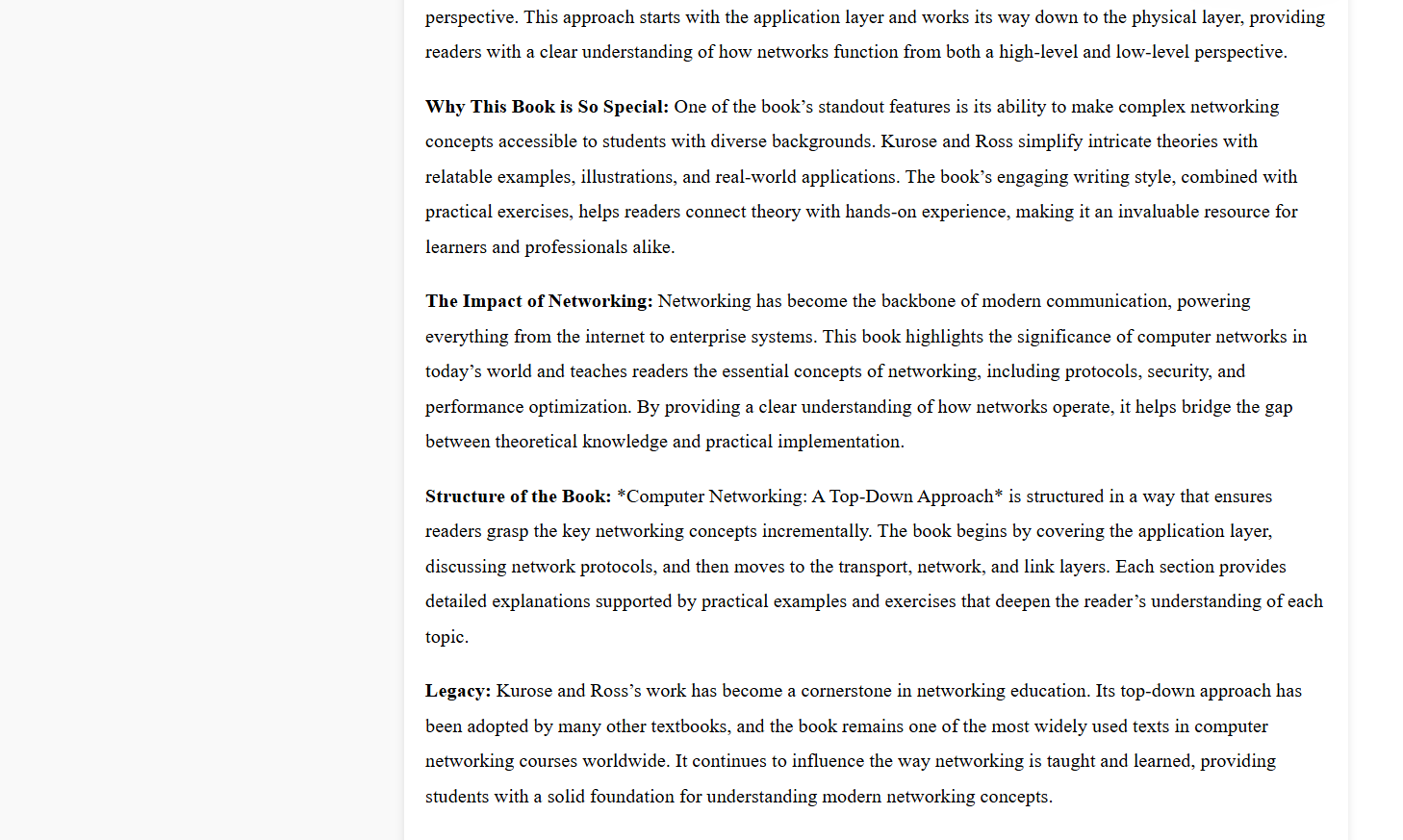
****

Source Code 5.2.2.5 - future-plans.php:

****

5.2.3 - Bestsellers & New Arrivals

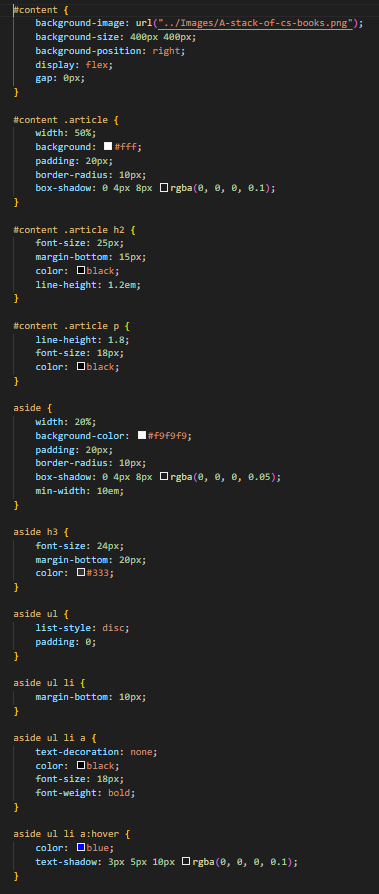
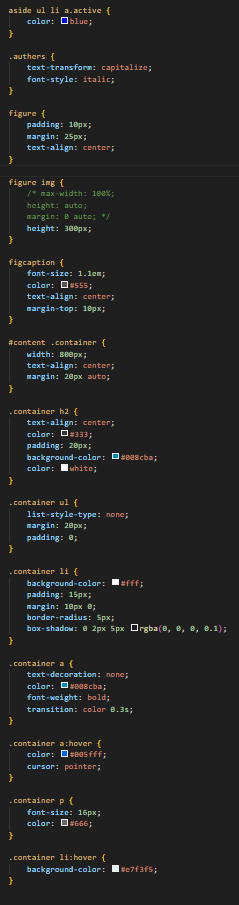
**Figure 5.2.3.1 – Bestsellers Homepage**

**One Sample Output Preview That Generalizes Other Books' Article Structure:**

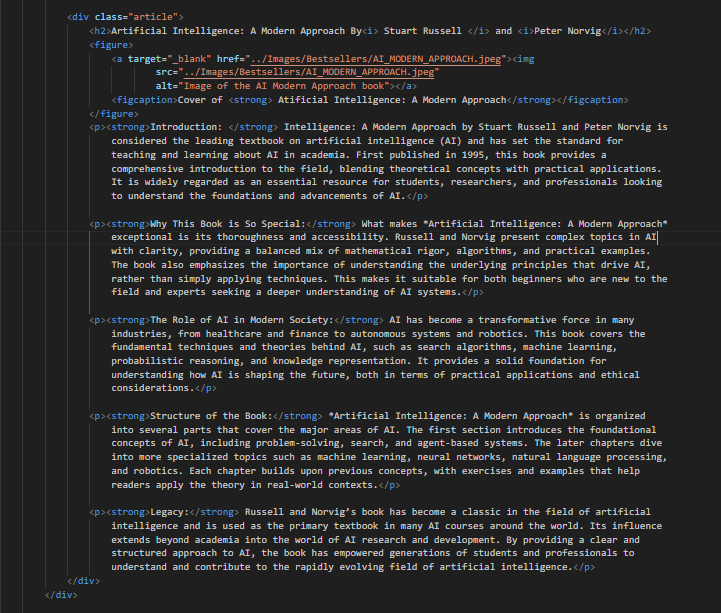
**Figure 5.2.3.2 – Article Section**

Source Code 5.2.3.1 - index.php

The homepage of the New Arrivals folder has the same webpage structure and appearance as of Bestsellers' homepage, but with different book articles. Therefore, it would be more convenient and time-saving if this file will be omitted.

Source Code 5.2.3.2 - bestseller.css

Source Code 5.2.3.3 - ai\_modern\_approach.php:

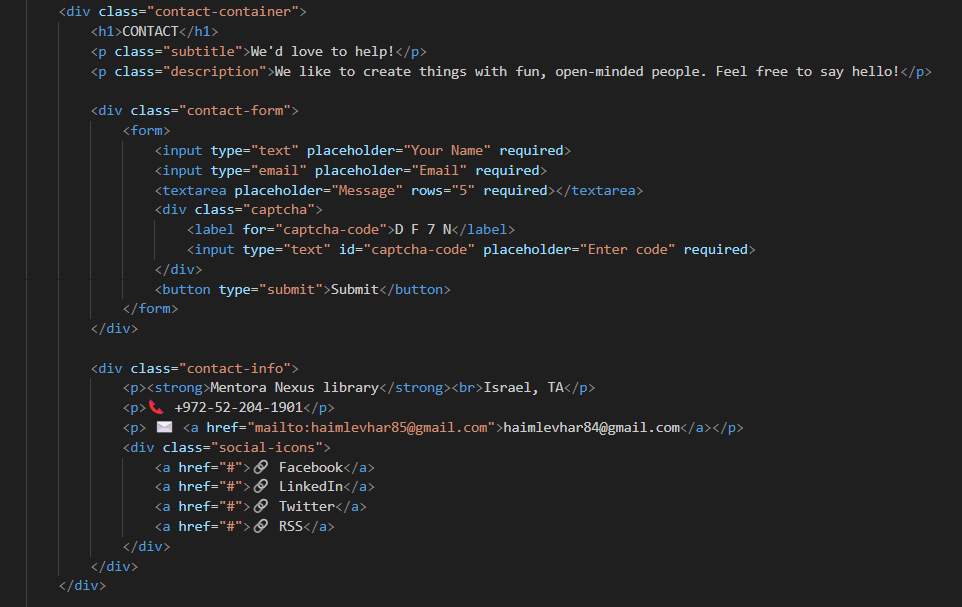


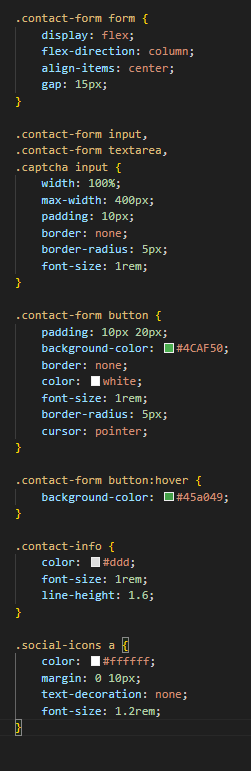
* Bestseller folder: c\_pro\_langu.php, clean\_code.php, com\_net.php, oper\_sys\_concep.php, pro\_langu\_prag.php
* New Arrivals folder: ai\_healthcare.php, brain\_tech\_cognition.php, cyber\_threat\_intelligence.php, data\_ai\_maintenance.php, metaheuristics\_iot.php

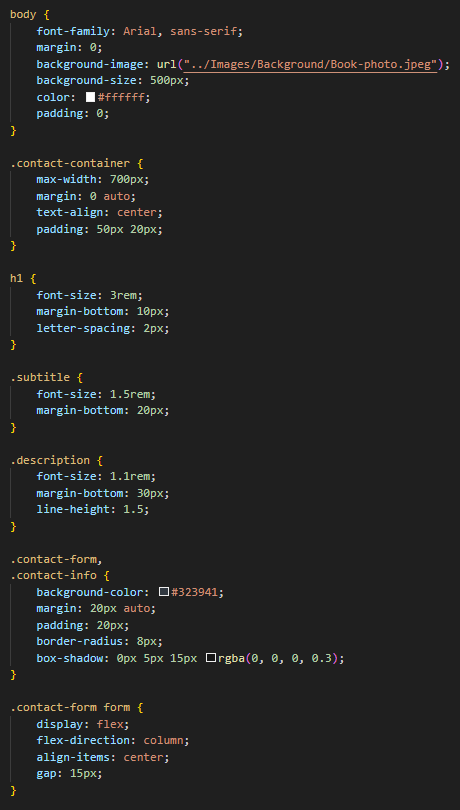
All the listed files have the same webpage structure and appearance as in the above html file but with different book articles. Therefore, it would be more convenient and time-saving if I exclude pictures of their code.

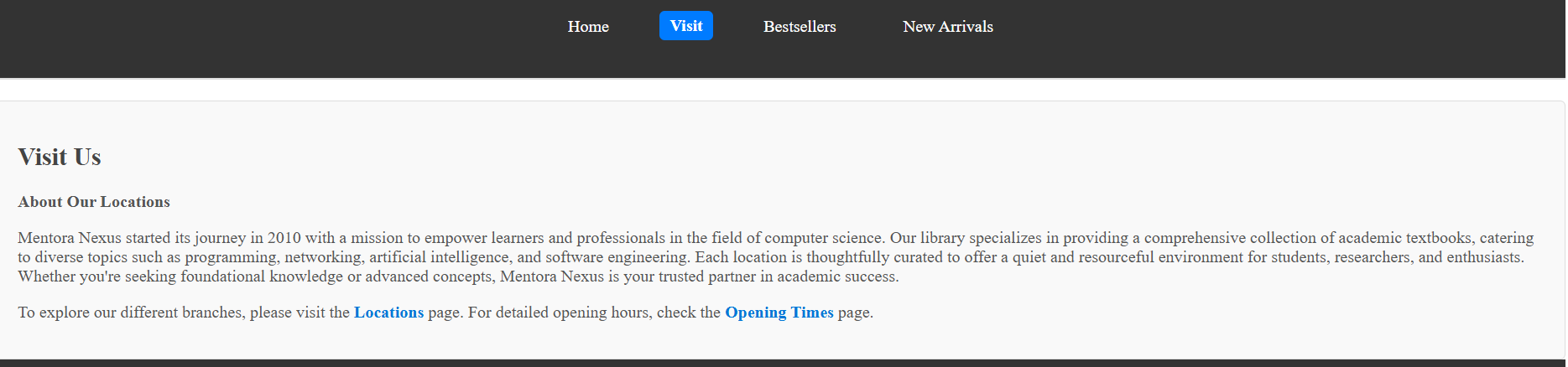
### 5.2.4 - Contact

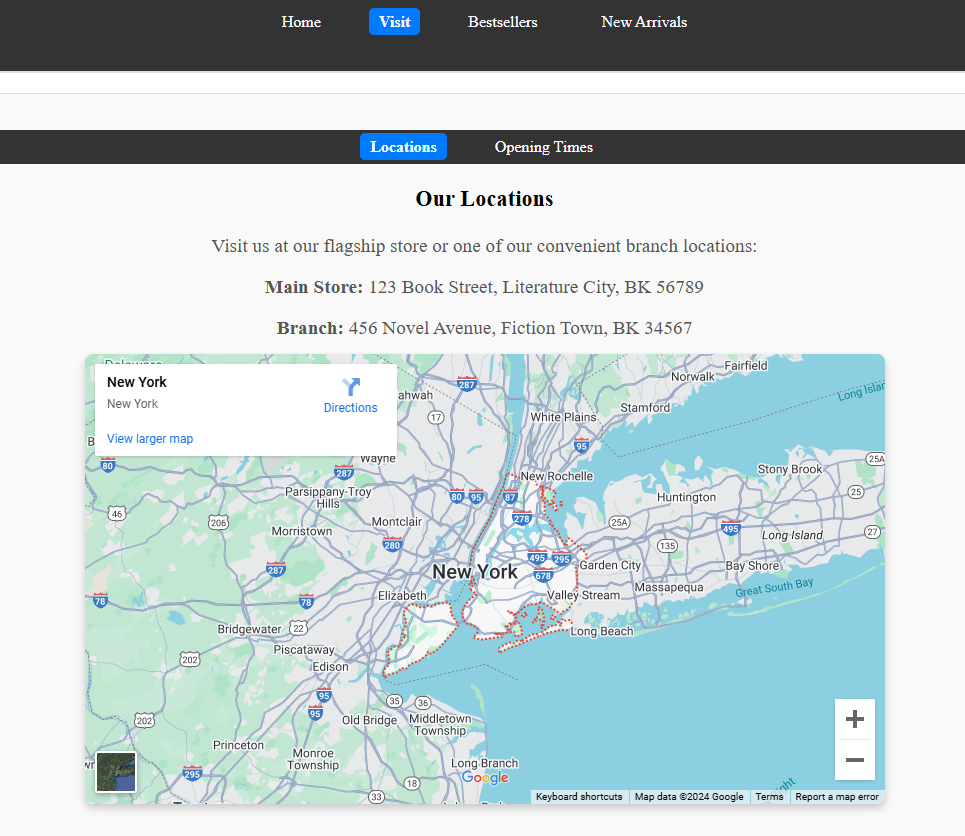
**Figure 5.2.4.1 - Contact Section**

****Source Code 5.2.4.1 - index.php:

Source Code 5.2.4.2 - contact.css:

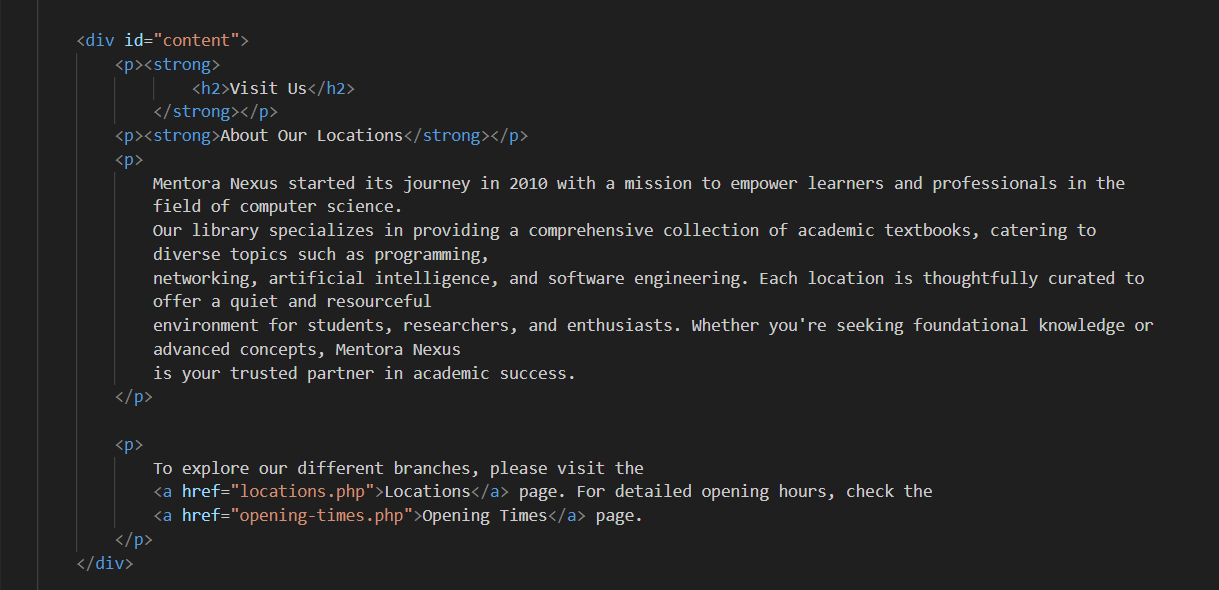


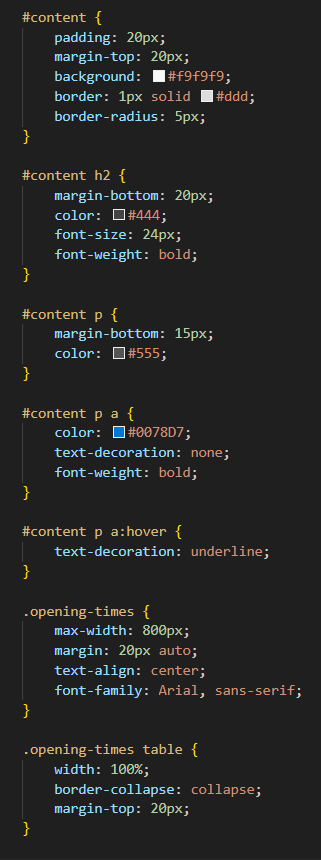
5.2.5 - Visit



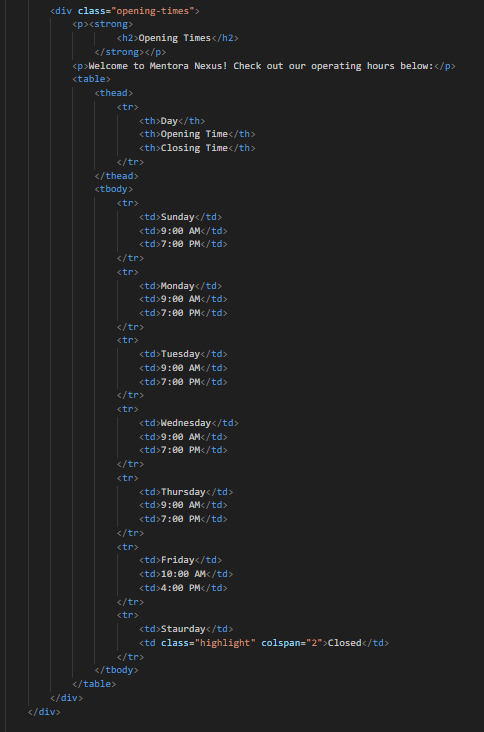


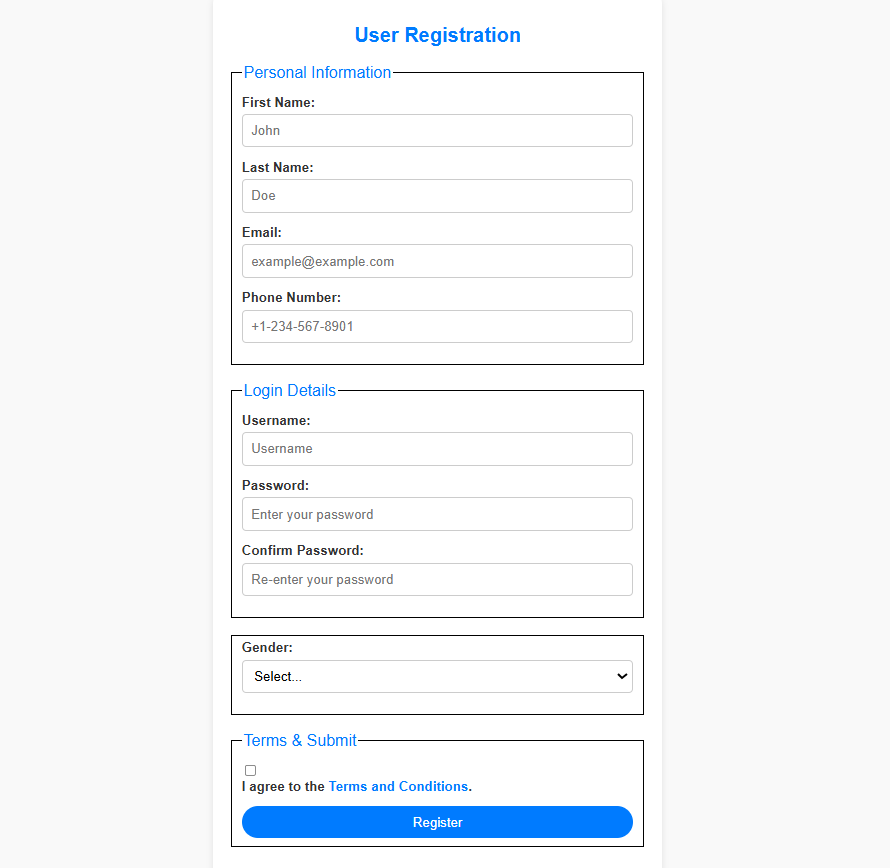
**Figure 5.2.5.1 – Visit Homepage, Locations, and Opening Times, respectively.**

Source Code ****5.2.5.1 – index.php:

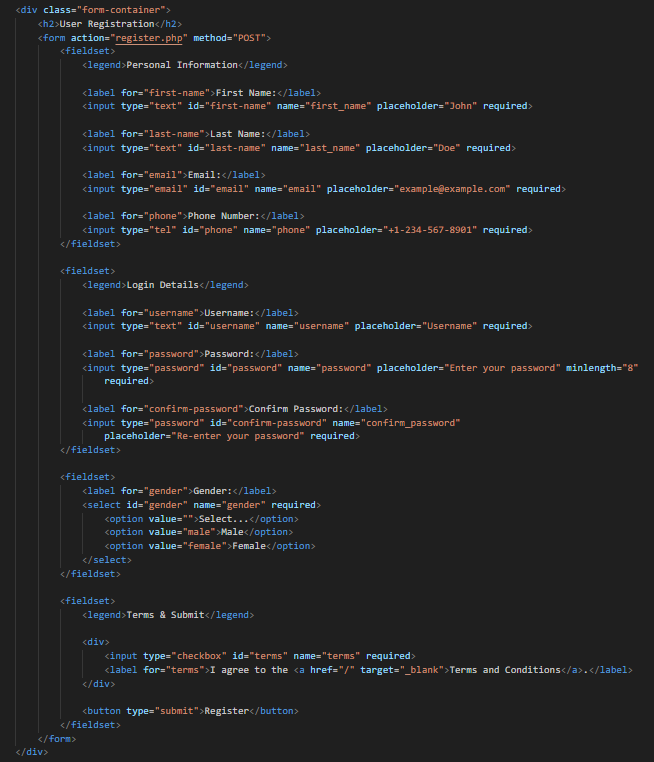
****Source Code 5.2.5.2 - visit.css:

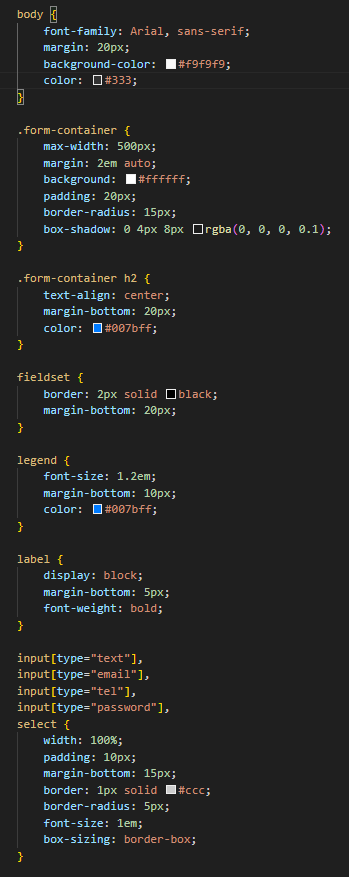
Source Code ****5.2.5.3 - locations.php:

****Source Code 5.2.5.4 - opening-times.php:

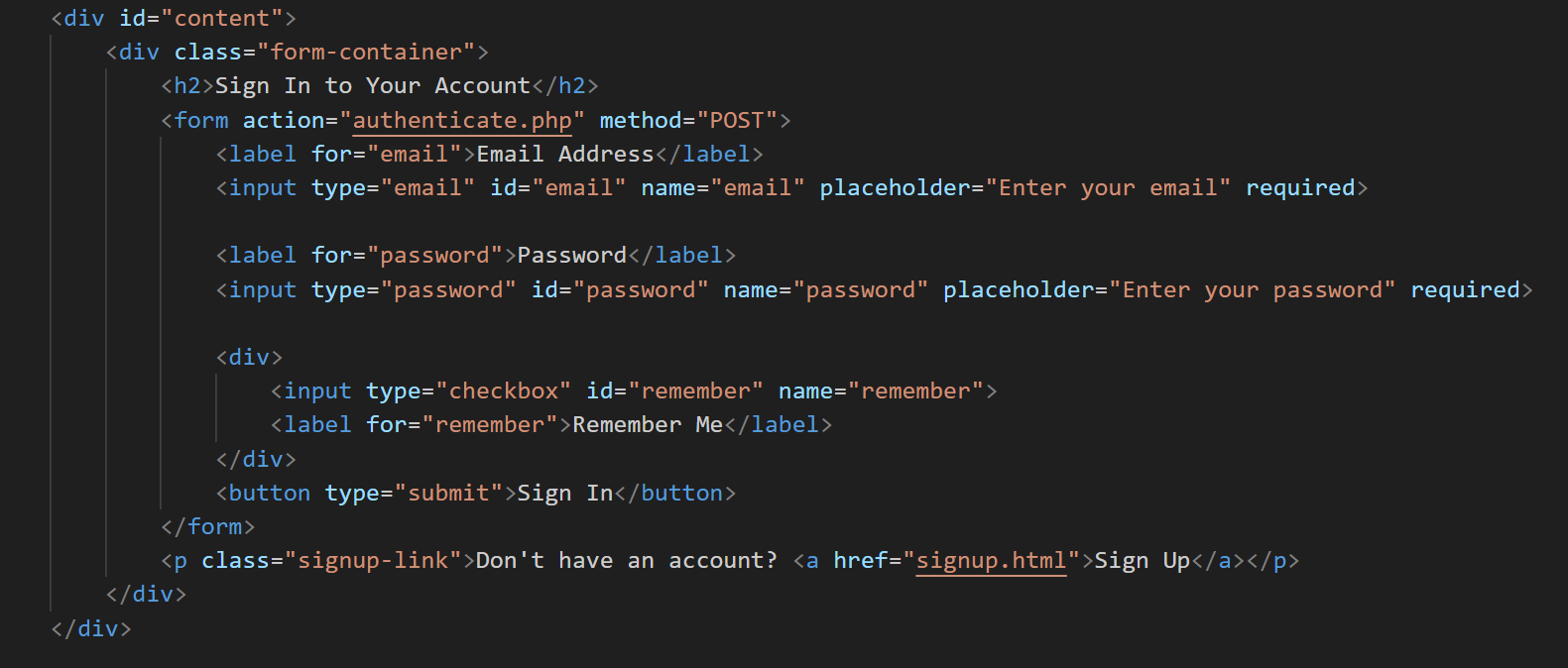
**** 5.2.6 - Users

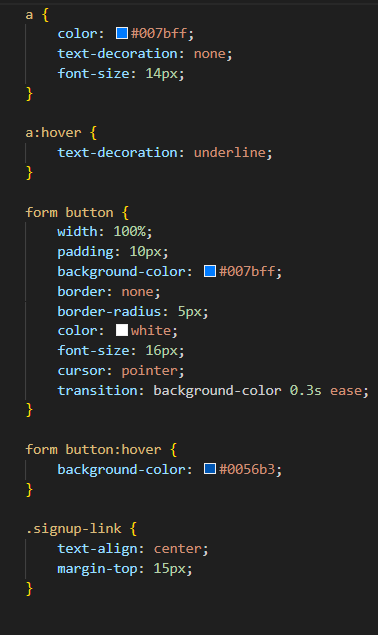
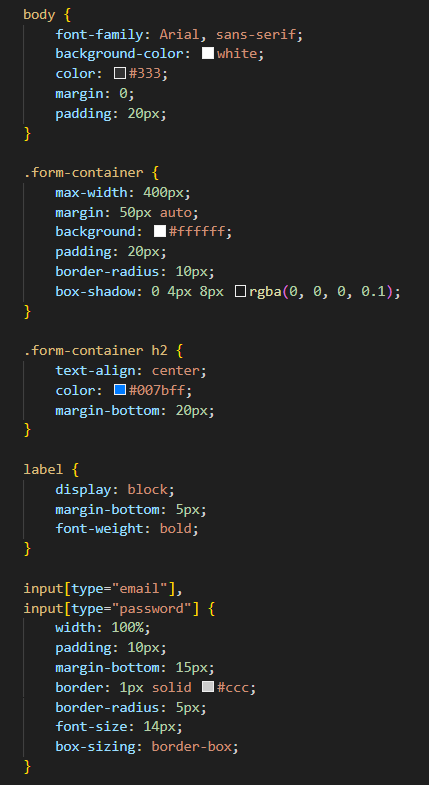
**Figure 5.2.6.1 - User Sign Up Form**

****Source Code 5.2.6.1 – signup.html:

Source Code ****5.2.6.2 – signup.css:

**Figure 5.2.6.2 – User Sign in Form**

Source Code 5.2.6.3 – signin.html:

Source Code ****5.2.6.4 – signin.css:

## 5.3 – PHP & JSON: User Registration & Validation

This section explains the processes of user registration and authentication for the Library Resource Portal, implemented using PHP and JSON. The forms allow users to sign up for an account and subsequently sign in to access personalized features. Visual outputs (refer to **Figure 5.2.10** for the signup form and **Figure 5.2.11** for the sign-in form) complement the source code (see **Source Code 5.3.1 - register.php** and **Source Code 5.3.2 - authenticate.php**) to illustrate how user inputs are validated and processed. The backend logic is designed to ensure seamless user interaction while maintaining data integrity and security.

### 5.3.1 - Description of the Process

#### **First-Time User: Signing Up:**

When a new user accesses the website, they are directed to the signup form (**Figure 5.2.10**) to create an account. The form collects essential credentials such as: email address, username, password, and additional optional details

The submitted information is processed by the PHP script in **register.php**. This script: validates the inputs to ensure they meet predefined criteria (e.g., valid email format, non-empty fields) and stores the validated data in a JSON file (**users.json – Source Code 5.3.4**) for secure and structured storage. Upon successful registration, a confirmation message is displayed, indicating the account creation is complete.

#### **Returning User: Signing In**

Users who have previously registered can log in via the sign-in form (**Figure 5.2.11**). This form requires: email and password.

The data is submitted to **authenticate.php**, which retrieves stored credentials from **users.json** and compares the provided information with the stored records.

If the credentials match, the user is redirected to the main homepage, with personalized content displayed. In this state, the headers replace the "Sign In/Sign Up" buttons with a "Logout" button (**Figure 5.3.1**), enhancing the user experience.

#### **Invalid Input Handling**

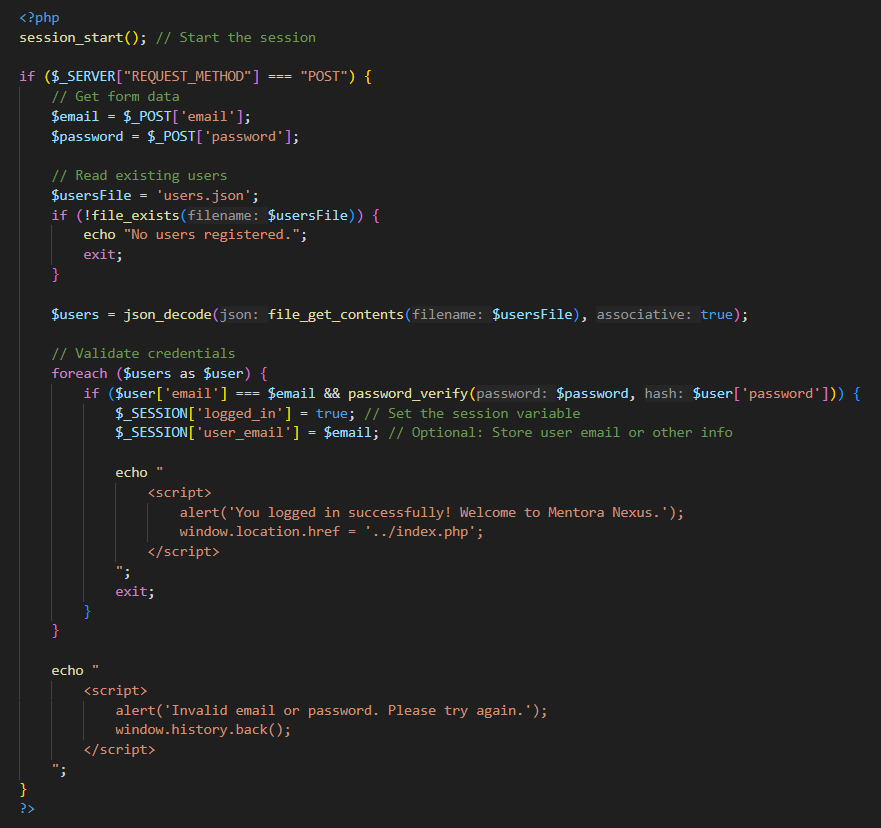
For both processes, invalid inputs trigger error messages, guiding the user to correct their entries. Examples include: a message indicating "User not found" for unregistered emails or prompt for "Incorrect password" when the password does not match the stored record.

#### **Logout Process**

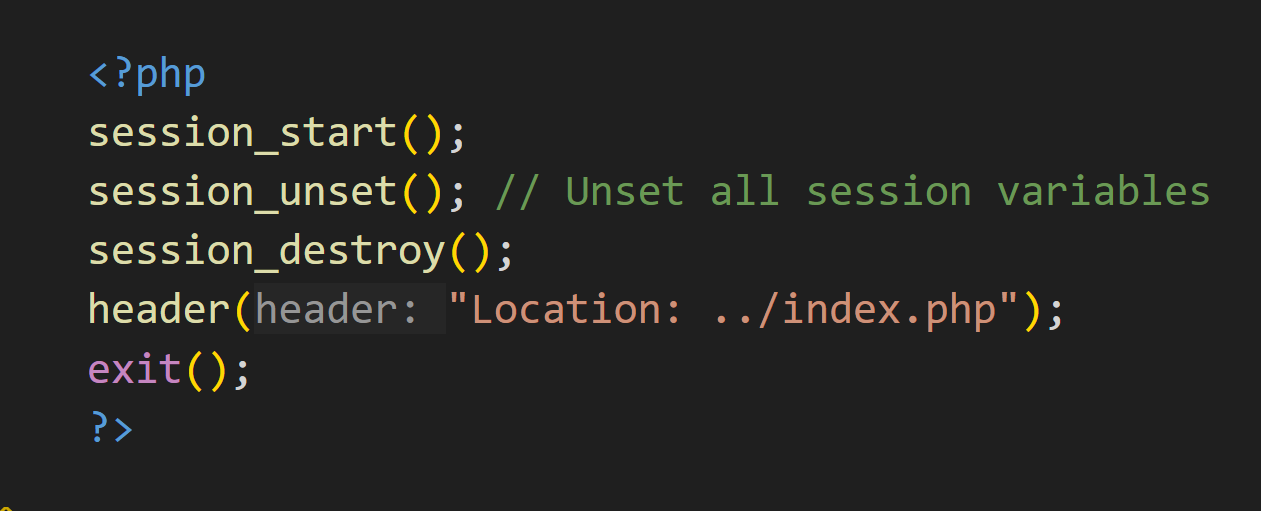
When a user clicks the "Logout" button (**Source Code 5.3.3 - logout.php**), the session terminates, and the user interface reverts to its initial state, displaying "Sign In/Sign Up" options.

This structured approach, combining dynamic PHP scripts and JSON storage, ensures an efficient and user-friendly account management system. The visual outputs of the forms and backend code snippets provide a comprehensive understanding of these processes.

****Source Code 5.3.1 – register.php:

****Source Code 5.3.2 – authenticate.php:

Source Code 5.3.3 – logout.php:

****

Source Code 5.3.4 - users.json:

****

**Figure 5.3.1 – Logout button replacing sign-in/up buttons.**

# HTML Elements Reference

In the reference below, I have provided HTML elements applied in this project alongside their categories and concise, complete descriptions. I have not provided code examples for these HTML elements, as these can usually be located via a search engine or even IDEs.

| **Tag** | **Category** | **Description** |
| --- | --- | --- |
| <html> | BLE | Groups the HTML code. |
| <head> | BLE | Contains information about the webpage. |
| <body> | BLE | Contains the main content of the webpage. |
| <title> | InlineE | A brief summarization of a few words that represents the webpage content. Its content - typically a short alphabetic string – is displayed on the webpage's tag. |
| <meta> | EE | Supplies a variety of metadata about the webpage. |
| <link> | EE | Links to an external style sheet. All CSS rules in this external file will be applied to their respective parts in the HTML file. |
| <div> | BLE | Groups a set of elements in one block-level box. |
| <a> | InlineE | Defines a hyperlink. Any text between the opening and closing <a> tag moves the user to the specified link. |
| <h1> to <h6> | BLE | Defines the font size of a text; the lower the number, the larger the text. |
| <p> | BLE | Defines a paragraph. |
| <ul> | BLE | Creates an unordered list with markers such as bullets, diamonds, discs, etc. |
| <ol> | BLE | Creates an ordered list with markers such as numbers, letters, or Romanic letters. |
| <li> | BLE | Defines a list item and is a child element of <ul> or <ol>. |
| <img> | InlineE, EE | Accepts a path to an image and displays it on the screen. |
| <form> | BLE | Groups form control elements. |
| <input> | BLE | Creates a form control based on the value of the type attribute. Text, making choices buttons, and submitting forms are all part of the type attribute. |
| <script> | BLE or InlineE | Contains JavaScript code. |
| <style> | BLE | Contains CSS style rules. Allows a CSS code to be directly within an HTML file. |
| <label> | InlineE | Describes the purpose of an input form, aiding vision-impaired users. This can warp descriptive text and a form control or associate itself with a form control using the for attribute with the form control's id. |
| <header> | BLE | Groups the header items such as navigation menus, search boxes, logos, sign-in/up forms, etc. It appears at the top of web pages. |
| <footer> | BLE | Groups the footer items such as copyright information, contact details, social media link, etc. It appears at the bottom of webpages. |
| <table> | BLE | Groups table elements. |
| <tr> | BLE | Creates a table row. |
| <th> | BLE, InlineE | Creates a table header, a child element of <tr> elements. |
| <td> | BLE, InlineE | Creates a table cell, a child element of <tr> elements. |
| <button> | InlineE | Allows for more control over button appearance. |
| <fieldset> | BLE | Groups related form controls. |
| <legend> | InlineE | Adds a caption to a <fieldset>. |
| <aside> | BLE | Groups information related to the currently displayed article but not essential to its overall meaning. |
| <iframe> | InlineE | Creates a window within the currently displayed window. |
| <figure> | BLE | Associates an image description written inside the <figcaption> element to the image inside the <figure> element. |
| <figcaption> | BLE | Associates an image description written inside the <figcaption> element to the image inside the <figure> element. |
| <select> | InlineE | Creates a drop-down list. |
| <option> | InlineE | Creates an option in a <select> drop-down list. |
| <textarea> | BLE | Creates a multiple-line text input. The number of lines is set by the *rows* attribute. EE |
| <br> | InlineE, EE | Inserts a new break. EE |
|  |  |  |

# CSS Properties Reference

In the reference below, I have provided CSS properties applied in this project alongside their concise, complete descriptions. I have not provided code examples for these CSS properties, as these can usually be located via a search engine or even IDEs.

| **Property** | **Description** |
| --- | --- |
| **line-height** | Specifies the vertical space between lines of text, often used to control text readability. |
| **width** | Defines the width of an element's content box. |
| **height** | Defines the height of an element's content box. |
| **margin** | Sets the space outside the element's border. It can accept up to four arguments (top, right, bottom, and left). |
| **border** | Specifies the size, style, and color of an element's border. |
| **background-color** | Defines the background color of the element. |
| **padding** | Specifies the space between the content of an element and its border. |
| **display** | Determines the type of rendering box for an element (e.g., block, inline, flex, grid, etc.). |
| **border-radius** | Rounds the corners of an element by the specified radius. |
| **cursor** | Specifies the type of cursor to display when pointing over an element. |
| **text-decoration** | Defines the decoration applied to text, such as underline, overline, line-through, or none. |
| **color** | Sets the color of text within the element. |
| **list-style** | Specifies the type, position, or image of list item markers. |
| **font-size** | Specifies the size of the text. |
| **letter-spacing** | Defines the spacing between letters in text. |
| **box-shadow** | Adds shadow effects to an element. Accepts values for horizontal offset, vertical offset, blur radius, spread radius, and optional color. |
| **font-family** | Specifies the font or typeface for text. |
| **text-align** | Aligns the text horizontally (e.g., left, right, center, justify). |
| **font-weight** | Sets the weight (thickness) of the font (e.g., normal, bold, or numerical values like 400, 700). |
| **align-items** | Aligns the items within a container along the cross-axis. |
| **justify-content** | Aligns items within a container along the main axis. |
| **transition** | Specifies the smooth change of CSS properties over a duration. |
| **gap** | Specifies the spacing between rows and/or columns in a layout. |
| **flex** | A shorthand for setting flex-grow, flex-shrink, and flex-basis. Used to define how a flex item grows, shrinks, and bases its size within a flex container. |
| **transform** | Applies transformations to an element such as rotate, scale, skew, or translate. |
| **:hover** | A pseudo-class that applies styles when the user hovers over the element. |
| **:active** | A pseudo-class that applies styles when an element is being activated by the user, such as when a mouse click is pressed but not yet released. |

# Index

A

**Attribute selectors** 7

C

**Cascading Style Sheets** 7

**Class selectors** 7

**client-side scripting language** 8

**CSS** *See* Cascading Style Sheet

**CSS selectors** 7

D

**Declarations** 7

**directory structure** 9

***<!DOCTYPE>* declaration 6**

**Dynamic websites** 8

E

**elements** 6

**external CSS files** 7

F

**form elements** 13

H

**HTML** 6

**HTML attributes** 6

**HTML files** 6

**HTTP** *See* HyperText Transfer Protocol

**HTTP requests** 9

**hypertext** 6

**HyperText Markup Language** 6

**Hypertext PreProcessor** 8

**Hypertext Transfer Protocol** 9

I

**ID selectors** 7

**Internet** 8

J

**JavaScript** 8

L

**Library Resource Portal** 9

M

**Markup languages** 6

O

**objects** 6

**opening tags** 6

P

**PHP** *See* Hypertext PreProcessor

**property name** 8

**property value**. 8

**Pseudo-classes** 7

S

**server-side scripting language** 8

**sitemap** 9

**source code** 13

**Static websites** 8

T

**tags** 6

**Tim Berners-Lee** 6

**Type selectors** 7

V

**validation scripts** 13

W

**web browser** 8

**web hosting company** 9

**webpage** 6, 7, 26, 28, 43