Final Project: E-Commerce Store

Overview

This Final Project will cumulate what you've been learning in the course with an opportunity to design, develop, and implement a full-stack website. You will be responsible for fully implementing both the front and back-end of an E-Commerce store of your choosing. Pick your theme, pick your features from the module component list, and have fun with it!

Please understand that this assignment is not a Creative Project. For full credit, this project must be clearly independent of your previous Creative Project (and HW) work. Ask the staff if you are unsure about similarity between previous assignments and your Final Project implementation.

Also, given the nature of this being a cumulative Final Project demonstrating the knowledge you have gained in this class, **you may not directly use code from online resources in this project**. You may still use external resources for inspiration/advanced features, but you should not be copy/pasting any code. As with CPs, you must explicitly cite any source you use so that we can know where it comes from (it is usually pretty obvious when code is taken from somewhere else, which would likely be considered plagiarism).

Whatever you create for this project is yours to share as you choose meaning you can publish it and/or show it off to others. That being said, we strongly encourage you to have fun and go (reasonably) above and beyond with this assignment. It is your chance to not only show off the valuable skills you've learned, but also develop a strong portfolio piece. Happy coding!

Learning Objectives

- Choose appropriate HTML tags with an understanding of the semantic meaning of each tag and the context they are used in a webpage.
- Use CSS properties to style a webpage to be in line with a pre-drafted Design Document.
- Modify your web page using JS and DOM objects. Produce readable and maintainable code with unobtrusive modular JS.
- Fetch plain text and/or JSON data from a web service using AJAX with fetch.
- Design and implement an API that responds to HTTP requests.
- Retrieve information from the server (e.g. with File I/O)
- Demonstrate understanding of full-stack code quality expectations modeled in lectures and the course's Code Quality Guide.
- Produce readable and maintainable code with clear documentation conforming to class standards.

Important Deadlines (Non-Seniors)

Below are the deadlines for *each* part of this Final Project.

Proposal due: Friday, May 28th 2AM

(Optional) Milestone due: June 3rd

Final Project due: June 10th

Reflection due: June 11th

Important Deadlines (Seniors)

The final day of classes for seniors is June 4th. Seniors will have a few modifications to support the earlier end of the term, including 1) CP4 being optional and 2) removing the requirement for "cart" features and only requiring 1 Chosen Feature instead of 2.

Proposal due: Friday, May 28th 2AM

(Optional) Milestone: You can schedule a 1-1 with Melissa to talk about your progress over Zoom before your final submission.

Final Project and Reflection due: June 4th

Starter Files and Deliverables

Proposal Requirements

The first submission for this Final Project will be the proposal. This will be submitted on Canvas, not on CodePost. Your milestone and final submission will be submitted on CodePost, just like all of the previous CPs and HW assignments in this course.

Proposal

Your proposal should include the items listed below:

- Wireframe of UI (2 or more "views" sketched out or done with an online tool, no need to be an artist). The wireframe should be labeled with text to include what each piece of the view does, either to the side or below the wireframe diagram(s).
    - Here are some examples of wireframes for different "views" of a hypothetical e-commerce store.
        - Example 1
        - Example 2

        - Example 3
        - Example 4
    - These wireframes are meant to help you get an idea of what your project is and help you design both your front-end and the needs of your back-end, and are also valuable to include in any web development portfolio. We suggest that each view have a number indicator representing a piece of the view. Each piece can then have a description of what the said piece is and does.
- You may find Figma and/or draw.io helpful for creating your wireframes.

All components of the proposal are due on **Friday, May 28th**. It will be reviewed by the staff so that you can receive feedback on scoping early on. You are not bound to your proposal meaning that if you change your mind on the structure of your E-Commerce store/color scheme/chosen feature points, you are welcome to change it. You should, however, have a solid idea of what you want to build. If you are feeling stuck refer to the bottom of this spec where we have listed some ideas introduced by the staff. You are welcome to use these ideas for your assignment.

We encourage you to submit prior to the due date if you have a comprehensive proposal and design document. This will allow you to move on to working on the milestone which you definitely could (and should) work on before receiving the feedback.

Milestone Requirements

We are also offering an optional milestone submission to provide feedback midway in the assignment on external/internal correctness. The milestone will not be graded, but we strongly encourage you to have the following requirements completed at this point.

At minimum, your milestone submission should contain, at a minimum, the .html and .css files for your E-Commerce website as well as the APIDOC. You are free to submit more, depending on how far along you are in the project, but having these parts done will help you get useful feedback before the final graded submission and avoid spending all of your time on the project in the final week. For this submission, you should aim to have your .html and .css files *at least* 75% completed meaning we should have a solid sense of the UI design. You are not bound to the files you submit and are welcome to change/update them prior to your final submission. However, you should have a solid idea of what E-Commerce store you are creating which should be reflected in this milestone submission.

Please ensure that all .html and .css files meet the code quality standards outlined in the Code Quality Guide. This includes, but is not limited to proper spacing, indentation, appropriate use of semantic tags and minimizing redundancy. Additionally, make sure that your code validates using the validators linked on the course website.

The APIDOC, which is the documentation for your API, should be complete with all the endpoints you expect to have. Please refer to the example documentation provided on Canvas for an example of expectations. You are not bound to the documentation you turn in for your

final submission but, given your proposal, you should have a strong idea of what endpoints will be required to achieve your proposed final product.

Final Submission Requirements

Your final submission should be turned in on CodePost with the files below:

File	Repository file you will implement
.html	at least one .html file
.css	at least one .css file
.js	At least one client-side .js file that will request the information from the server you've implemented and provide overall functionality to the page
.js	At least one .js file (e.g. app.js) representing the Node.js/Express web service for your E-Commerce site
.sql	(Optional) A .sql file that sets up the database/tables
APIDOC	The documentation file for your API

IMPORTANT NOTE: All static files (HTML/CSS/Client-side JS/any images) should be inside a "public" folder. All other files (e.g. your Node.js/Express web service) should be at the root, similar to the structure in lecture code/CP4. You will almost definitely want to use images for your products - you can create your own, or find some online but you must cite where your images came from at the bottom of your HTML page. If you're not sure where to start, you can find some good "icon packs" on flaticon.com which has a good overview of how to cite the icon sources for packs (you are free to use other images too). The Cafe website from lecture has a good example of citing public icons for use on a prototype store.

Any files that were submitted for the milestone should implement any feedback. Please do not include any files that are not relevant to your Final Project or include information in your APIDOC about an endpoint that is inaccurate and/or was not implemented.

External Requirements

Below are the external requirements for both the front- and back-end of your e-commerce store. For both the front- and back-end you are required to implement the "Required Features." In order for you to customize this project and make it your own we are allowing you to choose a few other features from the "Chosen Features". At minimum you are to choose and implement 2 of the chosen features (seniors only need to implement at least one). You are welcome and encouraged to add more features beyond this - this makes for a really strong project and a quality portfolio piece to showcase. In order to meet the requirements of this assignment your 2

"Chosen Features" should be from the ones listed below. However, if you think of a feature you would like to implement and is not on the list, you may request permission from Melissa to have that feature count as one of your two chosen features. Before requesting, note that in order to be considered for approval the requested feature must be one that requires both front- and back-end support (similarly to the other chosen features listed). The requested feature should not have a similar implementation to the other chosen features on the provided list.

Additionally, since you will be choosing and will be graded on your "Chosen Features" you will need to inform us of which ones you are choosing. You will do this when submitting the Final Project Proposal assignment on Canvas. Please note that you **will not** be eligible for full credit on this assignment if you do not select and implement two acceptable chosen features. This list ensures that there is no ambiguity and that the grader is clear on what features you chose to implement.

Disclaimer: You are welcome and encouraged to go above and beyond on this project. It is a chance for you to use and apply the skills you have acquired this quarter to a full stack project. That being said, if you do choose to go above and beyond you are responsible for ensuring all code meets our code quality guidelines. Any work you turn in for the final submission will be held to the same code quality expectations. If you are choosing to implement something not discussed in this course and are unsure if what you have is good code quality please reach out to the course staff.

Front End

- **Required Features**:
    - A way to see all of your service's products on a "main view" page
    - A way to see one product in a “single view” (at least 3 pieces of information such as the price/rating/name/colors/sizes it's available in etc.). You may choose to implement each "view" as a separate HTML page (e.g. index.html, contact.html, cart.html, etc.) or dynamically using JS/DOM manipulation.
        - The single view should be applicable and functioning for all products on your main view page.
    - A customer service view (e.g. contact page), which, at a minimum, must have a form for users to submit complaints/questions.
        - You must include some level of input validation (HTML5 or JS) to prevent invalid messages (such as empty messages) from being sent
            - This includes indications to the user (such as a message displayed on the screen) indicating that there is an issue with the form if all parts of the form are not filled out
            - You must also indicate to user that their message has been received
    - A way to filter through the products to only display ones with certain properties
        - This can either be accomplished with a search bar *or* a dropdown/other UI elements that, once selected, only display products matching the chosen query

- **(Non-Senior Requirement)** You must create a "cart" view which displays a list of items that the user is interested in
  - A user should be able to add/remove items from the cart
- You must have at least 10 CSS rule sets in your .css file and 10 unique rules **at minimum**
  - Must change at least 2 box model properties (border, padding, margin, height, width)
  - Must import and use at least one font from Google Fonts
  - Must use at least 2 flex properties (this excludes setting display: flex)
- Must use fetch to communicate with the back-end (your web service)
  - Errors should be gracefully handled and displayed to the user (you may not use alert, console.log or console.error). Consider your own experience on sites with error-handling.

Back End

- **Required Features**:
  - Must have at least one JSON response (including a plain text response is optional)
  - Must have a GET endpoint to request and return all of your service's products
  - Must have a GET endpoint to request and return information about a single item
    - This endpoint should be functional for each item in your e-commerce store
  - Must have a GET endpoint to request and return some of your service's products based on a filter (price, category, type etc.)
  - Must format the data from each request appropriately to send to the front-end (You should be able to explain why the way you designed your outputs are effective)
  - Must have a POST endpoint to accept and store all feedback received via customer service form (see front-end requirements)
  - Must have at least 2 files to store your data as txt or JSON (SQL tables are also sufficient if you have MySQL background)
  - Each file/table must contain at least 2 different attributes that are used to store or retrieve useful data
  - There should be at least 15 items in your E-commerce store (although we encourage you to add more to create a more interesting storefront).
    - (If you choose to use SQL, these items should be inserted into the table in a setup.sql file using INSERT statements so that we can populate a MySQL database on our end when testing - you will not receive full credit if we do not have a way to populate data)

Chosen Features

OPTION 1: An additional view indicating when/what promotions are available at your e-commerce store. There should be at least 5 promotions/sales available. All the information for the sales/promotions should be retrievable from a GET endpoint.

OPTION 2: A way for a user to buy a product/item from your e-commerce store. You should update the page indicating to the user the item has been sold and/or the order is on it's way. The updates to the stock of the item should be supported with a POST endpoint. Each time an item is bought the overall stock should be decremented.

OPTION 3: A way for a user to leave a review on an item. This review should be displayed on the page for other users to see and should be persistent across page reloads (using the appropriate storage technology). This feature should be supported with a POST endpoint so that each review left will be added to your file system or database.

OPTION 4: An additional view in your e-commerce store so that users have a way of becoming "loyal customers." This should be achieved with a form and should require at least three pieces of information from the customer (name, email, address, phone number etc.). Once a user joins successfully there should be a message displayed on the page indicating this. This should be supported with a POST endpoint so that all information on "loyal users" is added to your file system or database.

OPTION 5: A way for a user to customize a product. The user should be able to customize at least two features such as color of the product, material of the product etc. These changes should be supported by a POST endpoint which updates the item description with the new customizations.

OPTION 6: An additional view for a FAQ page. This should have at least 5 frequently asked questions and the answers displayed on the page for customers to read. This should be supported by a GET endpoint from which all the frequently asked questions and answers can be retrieved.

OPTION 7: An additional admin view allowing an administrator to add and/or remove items from the store as well as update current inventory. This should be supported with a POST endpoint which appropriately updates the file system or database with the correct information based on whether the admin has added/removed an item from the store.

OPTION 8: An additional admin view for managing the accounts that the e-commerce store users have created. An admin should be able to edit a user account and modify information. This should be supported with a POST endpoint which will appropriately update the user information based on the administrator's actions.

Internal Requirements

You are expected to follow the same Internal Requirements as the previous homework assignments you have completed. These include but are not limited to the following:

- **HTML/CSS**
  - Appropriate use of semantic tags
  - Consistent and readable spacing/indentation/etc.
  - Using classes/ids appropriately
  - Reducing redundancy in CSS; use descendent/group selectors to group together shared styles and do not have any overridden/unused styles
  - Appropriate file documentation
- **Client-side JS**
  - Must aim to minimize the use of module global variables unless strictly necessary. Prefer minimizing and localizing the scope.
    - Ex: DOM elements should never be stored as module globals as you *always* have access to them. Variables only used in one function should not be stored as module globals as it is completely unnecessary
  - Must use functions to minimize redundancy in your code
  - Must not have "do all" functions - instead break down into several functions such that each has a specific purpose (functions more than 30 lines should likely be factored out with helper functions)
  - You should not make any unnecessary fetch requests
    - E.g. making a request before the information is required or if the given information is already available
  - You must have appropriate documentation for all functions/files.
- **Server-side JS**
  - Should use async/await appropriately
  - Choose correctly between the type of request (GET vs POST)
  - Should not override content headers (e.g. using res.type(“text”/”json”) twice in one request-response cycle)
  - Should set error types appropriately
- **File I/O or SQL**
  - (File I/O Option) Must define appropriate file structure (e.g. .txt, .json, images, etc.)
  - (File I/O Option) Should properly use functions from fs/promises module without unnecessary processing of files
  - (SQL Option) Must define appropriate data types for each column in each table
  - (SQL Option) Must have appropriate documentation
    - When documenting your SQL file, a brief header comment is sufficient with your student information and what the database/table(s) represent.

Reflection Requirements

The reflection for this Final Project will be due on June 11th. This document should be at least 1 page (double spaced, 12pt font, Times New Roman) and should be a reflection of your experience with the implementation process. Questions you could address are:

- What were the hardest features to implement? How did you approach these problems?
- What resources were most helpful to you (either provided by the course or found online)?
- Were there features you tried to implement and were unsuccessful? What were they and what part of the implementation stumped you?
- If you had another week to work on this project, what would you add to your nifty store?
- What did you enjoy the most about this project? What was the most rewarding?
- Was the breakdown and checkpoints throughout the assignment helpful? Would you change how the assignment is broken down/what is due?

You can answer all of these questions, some of them or none. What we care about most is that you are reflecting on your experiences working on a full-stack project and evaluating your successes/failures along the way.

The reflection should either be a word document (.doc) or a PDF (.pdf). No other submission formats will be accepted. Similarly to the Proposal and Design Document **you will submit your reflection on Canvas** and not through CodePost.

Development Strategies

Given the size of this project, you may find it hard to get started or know how to decide how to break down a problem. Below are some development strategies that may be helpful if you are unsure of how to proceed at some point of your development process. Don't be content with staying stuck. If, at any point in the development process you feel stuck, please be proactive about getting help. These tips will hopefully help:

- Brainstorming: Before even starting on the wireframe, it can help to just dump all of your ideas onto a piece of scratch paper (a text document works as well, but having the freedom to sketch out anything that comes to mind certainly supports the creative process).
    - Additionally, you can quickly draft some rough outlines for your site, and quickly gauge what works/doesn't work
- When reading the spec, highlight the parts and make a checklist of everything you see that is specific and needs to be accomplished and then check things off as you go.
- Before jumping into any HTML/CSS, draw a wireframe to outline your page idea, just starting with boxes (for containers) and lines (for text). Try to do this with a few drafts, adding a few more design elements each time. If you get stuck with layout/design ideas, go to other websites for inspiration (e.g. AWWards has a great showcase of different website designs, and you can never go wrong with CSSZenGarden)

- For an APIDOC, it's almost always best to try to write this before starting your server-side code. What information do you *need* from the service? What kind of parameters are good to have to filter a request? How can you represent the response data in JSON cleanly and in a way that is understandable to a client? What are the possible erroneous requests a client might make that your service should respond to with a useful error message?
- Talk to Melissa/TAs about your design process/brainstorming if you're having trouble getting started. Often you can get unexpected inspiration just by talking about different ideas, goals, and challenges you're having when planning a new project!
- Develop and test the API separately from the JS. Once it's done, you can just plug it into the JS and know it works. You can use Postman to test it.
- When developing the backend, it's a really good idea to include test code that populates your files and/or database with fake data so that you can easily test the output of each end-point of your API.
- Make use of the resources you have. Post to Discord with questions, come to OH, talk to the staff, use the lecture material we have provided.
- You may find it useful to draw diagrams if you are not sure where to start in terms of code. Think of what your problem is and what pieces you need in order to solve it. Drawing arrows/maps connecting the pieces can be helpful.

Grading Breakdown

This assignment, in total, will be out of 30 points and is worth 20% of the course grade as noted on the course syllabus. The key areas we will be looking at directly relate to the listed learning objectives, your ability to meet the specifications for the external behavior, and the internal correctness of your code. A potential rubric might be summarized as:

1. **Proposal:** - 4pt
2. **Checkpoint:** (.html file(s), .css file(s), APIDOC) - optional
3. **External Correctness:** - 13pt
4. **Internal Correctness:** - 8pt
5. **Documentation** - 2pt
6. **Reflection:** - 2pt

Credit: Special thanks to Tal Wolman for helping with the original version of this project (2019).