Frontend Engineering Curriculum

- Learn HTML & CSS (2 weeks)
  - HTML & CSS Crash Course Tutorial - YouTube
  - CSS Gradients and repeating gradients - YouTube
  - How to Add Google Font to HTML Website - YouTube
  - How to Use Font Awesome Icons on HTML Website - YouTube
- **Build a product/service landing page (2 weeks)**
  - Details:
    1. This should be a single-page website to promote and sell a specific product/service (e.g., a headset), similar to this page: iPhone 14 Pro and iPhone 14 Pro Max - Apple but shorter.
    2. Include a compelling headline, product/service description, key features, pricing, and a call to action e.g., a "buy now" button (clicking on the button should do nothing).
    3. Use fonts from Google Fonts and social media icons from Font Awesome.
    4. It should be responsive to different screen sizes.
  - Tips:
    1. Google search or visit Dribbble for UI (user interface) ideas.
    2. Get colour palette ideas from Flat UI Colors (or elsewhere).
    3. Get images from Pexels (or elsewhere).
    4. Use ChatGPT to:
       - Get answers to any questions you might have.
       - Learn about anything e.g., an HTML tag, a CSS property, etc.
       - Find problems in your code.
       - Get ideas.
       - And so on.
- Learn more CSS (2 weeks)
  - CSS Flexbox Tutorial - YouTube
  - Build Layouts with CSS Grid - YouTube
  - CSS Animation Tutorial - YouTube
- **Build a website for a business of your choice (2 weeks)**
  - Homepage:
    1. Design the homepage layout using Flexbox.
    2. Create a header section with a logo, navigation menu, and social media icons aligned horizontally using Flexbox.
    3. Implement a hero section with an attractive image and a call-to-action button centred using Flexbox.
    4. Include a section to showcase featured products/services using Flexbox to create a grid or row-based layout.
  - About Us Page:
    1. Create a new page for the "About Us" section.
    2. Implement a header and footer that are consistent with the homepage.
    3. Design the main content section using Flexbox.
    4. Include text content about the business, its history, mission, and values.

    - Products/Services Page:
        1. Create a dedicated page to showcase various products/services.
        2. Implement a header and footer that are consistent with the other pages.
        3. Design the product/service listing section using Flexbox to create a grid or row-based layout.
        4. Include product/service images, titles, descriptions, and prices using Flexbox for alignment and spacing.
    - Contact Page:
        1. Create a separate page for the contact information and a contact form.
        2. Implement a header and footer that are consistent with the other pages.
        3. Design the contact information section using Flexbox for alignment.
        4. Include the business address, phone number, and email using Flexbox.
        5. Implement a contact form using Flexbox to align the input fields, labels, and submit button.
    - Navigation and Responsive Design:
        1. Ensure that the website's navigation menu is consistent across all pages.
        2. Use Flexbox to create a responsive navigation menu that adjusts well on different screen sizes.
        3. Implement responsive design techniques using media queries and Flexbox properties like flex-wrap and flex-direction to make the website mobile-friendly.
    - Styling and Visual Appeal:
        1. Add hover effects and animations to create an interactive experience.
        2. Test the website's layout and responsiveness on various devices and screen sizes.
- Learn Git and GitHub (1 week)
    - Git & GitHub Tutorial for Beginners - YouTube
    - Gitflow Workflow | Atlassian Git Tutorial
    - How to Write Better Git Commit Messages – A Step-By-Step Guide
- **Push your previous projects to GitHub (1 day)**
    - Always use Git while working on subsequent projects.
    - Host the websites you built with GitHub Pages.
        1. Getting Started with GitHub Pages - YouTube
- Learn JavaScript (3 weeks)
    - Modern JavaScript Tutorial - YouTube
    - JavaScript DOM Tutorial - YouTube
- **Build a Random Quote Generator (1 week)**
    - HTML Structure:
        1. Create a basic HTML structure to display the quote.
        2. Add a button element that users can click to generate a new quote.
    - Quotes Data:
        1. Create an array of objects to store a collection of quotes.
        2. Each quote should have a "quote" property for the actual quote and an "author" property.
    - JavaScript Logic:
        1. Write JavaScript code to select a random quote from the quotes data when the button is clicked.
        2. Use the Math.random() function to generate a random index within the range of the quotes array length.

        iii. Access the randomly selected quote from the array and display it on the page.
    - Displaying the Quote:
        i. Use JavaScript to manipulate the HTML element that will display the quote and author.
        ii. Update the text content of the element to show the randomly selected quote.
        iii. Include a feature to automatically generate a new quote after a certain time interval. Check out Scheduling: setTimeout and setInterval.
- Learn more JavaScript (4 weeks)
    - Learn JavaScript | Codecademy
    - Learn Intermediate JavaScript | Codecademy
    - Object Oriented JavaScript - YouTube
    - Tip: You can read about specific topics at https://javascript.info/.
- **Build an Inventory Tracker (2 weeks)**
    - Description: An inventory tracker is a tool used to monitor and manage the stock or inventory of products or items in a business or personal setting. It helps keep track of the quantities, availability, and details of various items, ensuring efficient inventory management.
    - Create Product Class:
        i. Create a Product class with properties such as name, quantity, description, and any other relevant attributes.
        ii. Implement a constructor method to initialize the product object with the provided values.
        iii. Add methods to the class, such as getters and setters for accessing and modifying the product's properties.
    - Create Inventory Class:
        i. Create an Inventory class that will manage the collection of products.
        ii. Implement an array or object property to store the products.
        iii. Add methods to the class for adding products, updating quantities, and deleting products from the inventory.
    - HTML Structure:
        i. Create a basic HTML structure with input fields, buttons, and a table to display the inventory.
        ii. Use appropriate HTML elements, such as <input>, <button>, and <table>.
    - Instantiate Inventory Object:
        i. Instantiate an instance of the Inventory class to represent the inventory tracker.
        ii. This object will be used to manage the inventory data and perform operations on it.
    - Add Product Functionality:
        i. Implement JavaScript code to handle adding products to the inventory.
        ii. Create an event listener on the "Add" button to capture user input from the input fields.
        iii. Instantiate a new Product object with the input values.
        iv. Use the Inventory object to add the newly created product to the inventory data structure.
        v. Update the table in the HTML to display the newly added product.
    - Display Inventory:
        i. Write JavaScript code to display the products in a table format.
        ii. Use the Inventory object to retrieve the inventory data.
        iii. Loop through the products and dynamically create table rows and cells to represent each product.

        iv. Populate the cells with the corresponding product details retrieved from the Product objects.
    - Update Quantity Functionality:
        i. Implement JavaScript code to handle updating the quantity of products.
        ii. Add event listeners to the quantity input fields to capture user changes.
        iii. Retrieve the corresponding Product object from the inventory using its unique identifier (e.g., product ID).
        iv. Use the Inventory object's methods to update the quantity of the product.
        v. Reflect the updated quantity in the corresponding table cell.
    - Delete Item Functionality:
        i. Write JavaScript code to handle removing products from the inventory.
        ii. Add event listeners to the delete buttons associated with each product.
        iii. Retrieve the corresponding Product object from the inventory using its unique identifier.
        iv. Use the Inventory object's methods to remove the product from the inventory data structure.
        v. Update the HTML table accordingly.
- Learn Asynchronous JavaScript (1 week)
    - Asynchronous JavaScript (2020 version) - YouTube
    - Using the Fetch API
- **Build a Random Image Feed (2 weeks)**
    - API Details:
        i. Visit the Lorem Picsum API website and review the available options and endpoints.
        ii. Understand how to construct the URL to retrieve a list of images.
    - HTML Structure:
        i. Create an HTML structure (similar to Instagram) to display the retrieved images.
        ii. Include a container element for the images and a loading indicator.
    - API Integration:
        i. Write JavaScript code to integrate with the Lorem Picsum API using the fetch function.
        ii. Construct the URL for fetching images with the desired sizes and options. Fetch ten images per request.
    - Handle API Response:
        i. Retrieve the response from the API using the fetch function.
        ii. Extract the image URLs or relevant data from the response.
    - Display Initial Images:
        i. Use JavaScript to update the HTML with the retrieved image URLs.
        ii. Create HTML elements dynamically with the initial images.
    - Implement Infinite Scroll:
        i. Add a scroll event listener to detect when the user reaches the bottom of the page.
        ii. When the bottom is reached, trigger a function to fetch and append more images.
    - Fetch More Images:
        i. Modify the API request to fetch a new set of images.
        ii. Append the new images to the existing container element in the HTML.
    - Loading Indicator:

        i. Display a loading indicator at the bottom of the page when new images are being fetched. Check out [CSS Loaders Tutorial - YouTube]().
    - Error Handling:
        i. Implement error handling to handle cases where the API request fails or returns an error status.
        ii. Display appropriate error messages to the user in case of errors.
- Learn Data Storage in the Browser (1 week)
    - [LocalStorage, sessionStorage]()
    - [Learn localStorage in JavaScript by building a project! - YouTube]()
- **Build a Bookmark Manager (2 weeks)**
    - Description: A bookmark manager is a tool that helps users organize and keep track of their favourite websites or web pages.
    - HTML Structure:
        i. Create an HTML structure for the bookmark manager's user interface.
        ii. Include input fields for the bookmark title and URL, buttons for adding/removing/editing bookmarks, and a container for displaying the bookmarks.
    - Create Functions:
        i. Write JavaScript functions to handle various aspects of the bookmark manager.
        ii. Create functions for adding a bookmark (including title and URL), removing a bookmark, editing a bookmark, and updating the display.
    - Store Data in localStorage:
        i. Utilize the localStorage API to store the bookmark data.
        ii. When adding, removing, or editing a bookmark, update the localStorage accordingly.
    - Retrieve Data from localStorage:
        i. Retrieve the bookmark data from localStorage when the application loads.
        ii. Display the stored bookmarks (including titles and URLs) in the application's interface.
    - User Interaction:
        i. Add event listeners to buttons and input fields to capture user input.
        ii. Implement functionality to add new bookmarks (including titles and URLs), remove existing bookmarks, and edit existing bookmarks.
    - Update Display:
        i. Write code to update the display whenever a bookmark is added, removed, or edited.
        ii. Use JavaScript to manipulate the DOM and reflect the changes in the bookmark display.
    - Error Handling:
        i. Implement error handling to handle cases where localStorage is not available or encounters errors.
        ii. Provide appropriate error messages or fallback behaviour.
- Learn about computers and the Internet (1 week)
    - [Unit: Computers and the Internet - Code.org]()
    - [How the Internet Works for Developers - Pt 1 - Overview & Frontend]()
- Read [Eloquent JavaScript]() (4 weeks)
- Learn React (3 weeks)
    - [Full Modern React Tutorial - YouTube]()
    - [React]()

- **Build a Note-Taking application (2 weeks)**
    - Component Hierarchy:
        1. Plan the component hierarchy for your note-taking application.
        2. Identify the main components needed, such as a NotesList component, NoteForm component, and NoteItem component.
    - Create Components:
        1. Create the necessary components using functional components in React.
        2. Set up the basic structure of each component with JSX.
        3. Include an input field for the note title, a text area for the note content and buttons for adding/removing/editing a note.
    - State Management:
        1. Implement state management using React's useState or useReducer hooks.
        2. Set up state variables to manage the notes data, such as an array of notes.
    - Initialize Notes from localStorage:
        1. Check if there are any existing notes in localStorage.
        2. If notes exist, retrieve them from localStorage and initialize the state with the retrieved data.
    - Display Notes:
        1. Render the notes data in the NotesList component by mapping over the notes array.
        2. Display each note using the NoteItem component and pass the necessary props.
    - Add Note Functionality:
        1. Implement functionality to add a new note using the NoteForm component.
        2. Capture user input and update the notes state with the new note.
        3. Update the localStorage with the updated notes data.
    - Remove Note Functionality:
        1. Implement functionality to remove a note from the notes state.
        2. Provide a way to delete a specific note, such as a delete button in the NoteItem component.
        3. Update the localStorage with the updated notes data.
    - Edit Note Functionality:
        1. Provide a way to edit existing notes, such as a button or double-clicking on a note.
        2. When editing a note, display the note's text in an editable text area.
        3. Update the notes state and localStorage with the edited note.
    - Data Persistence with localStorage:
        1. Update the notes state in React whenever a note is added, removed, or edited.
        2. Utilize the localStorage API to persist the notes data across page refreshes.
        3. Update the localStorage with the latest notes data whenever there is a state change.
- Learn Redux (1 week)
- Learn React #12: Redux & React Intro - Redux Crash Course
    - Redux Tutorial - Learn Redux from Scratch
- **Build a Job Application Tracker (2 weeks)**
    - Description: A Job Application Tracker allows users to track and manage their job applications in a centralized system. It provides a user-friendly interface where users can add, view, edit and delete job

applications, along with relevant details such as job titles, company names, application statuses, and application dates.

- Component Hierarchy:
    1. Plan the component hierarchy for your Job Application Tracker.
    2. Identify the main components needed, such as JobList, JobForm, and JobItem components.
- Create Redux Store:
    1. Set up a Redux store to manage the application's state.
    2. Define actions and reducers for managing job application data.
- Create Components:
    1. Create the necessary components using functional or class components in React.
    2. Set up the basic structure of each component with JSX.
- Connect Components to Redux Store:
    1. Use the connect function from react-redux to connect components to the Redux store.
    2. Map state and dispatch functions to component props.
- Initialize Job Applications from localStorage:
    1. Check if there are any existing job applications in localStorage.
    2. If job applications exist, retrieve them from localStorage and initialize the Redux store with the retrieved data.
- Display Job Applications:
    1. Render the job applications from the Redux store in the JobList component.
    2. Display each job application using the JobItem component and pass the necessary props.
- Add Job Application Functionality:
    1. Implement functionality to add a new job application using the JobForm component.
    2. Capture user input and dispatch an action to add the new job application to the Redux store.
    3. Update the localStorage with the updated job application data.
- Remove Job Application Functionality:
    1. Implement functionality to remove a job application from the Redux store.
    2. Provide a way to delete a specific job application, such as a delete button in the JobItem component.
    3. Update the localStorage with the updated job application data.
- Edit Job Application Functionality:
    1. Implement functionality to edit a job application.
    2. Provide a way to edit specific fields of a job application, such as a form within the JobItem component.
    3. Dispatch an action to update the Redux store with the edited job application data.
    4. Update the localStorage with the updated job application data.
- Data Persistence with localStorage:
    1. Update the Redux store whenever a job application is added, removed, or edited.
    2. Utilize the localStorage API to persist the job application data across page refreshes.
    3. Update the localStorage with the latest job application data whenever there is a state change.
- Styling and Layout:
    1. Use Tailwind CSS to style the components.