**Summary of key concepts**

**HTML**

1. **HTML Structure:**
   * The HTML file begins with the **<!DOCTYPE html>** declaration, specifying the document type and language.
   * The document contains a **<head>** section with metadata, including character set, viewport settings, and links to external stylesheets (Font Awesome and a custom "style.css").
   * The **<body>** section holds the main content of the page.
2. **Page Structure:**
   * The content is organized within a **<div class="container">**, which contains a header, a form for adding items, a filter input, a list of items, and a "Clear All" button.
   * The header includes an image (icon) and a title.
3. **External Dependencies:**
   * The page relies on Font Awesome for icons. It imports the Font Awesome stylesheet from a CDN.
4. **Form for Adding Items:**
   * There's a form with an input field (**<input type="text" id="item-input" />**) for entering items and a submit button.
   * The form has an ID (**"item-form"**) for potential JavaScript interaction.
5. **Filter Input:**
   * There's an input field (**<input type="text" id="filter" />**) for filtering items.
6. **List of Items:**
   * The list is represented by an unordered list (**<ul id="item-list">**). Currently, it contains commented-out list items as examples.
   * Each list item may have a name (e.g., "Apples") and a button for removing the item.
7. **JavaScript Integration:**
   * The page includes a JavaScript file (**<script src="app.js"></script>**), suggesting dynamic functionality.
   * The JavaScript file (**app.js**) likely handles interactions such as adding, removing items, and clearing the list.
8. **Styling:**
   * Styling is separated into an external stylesheet (**style.css**), linked in the **<head>** section.
9. **Accessibility:**
   * Appropriate meta tags and attributes are used for better accessibility and compatibility (**charset**, **viewport**, etc.).

In summary, this code sets up a basic structure for a shopping list application with HTML, styles it with CSS, and potentially adds dynamic functionality using JavaScript. The focus is on user interaction, allowing users to add, remove, and filter items on their shopping list.

**CSS**

1. **Font Styles:**
   * The code imports the 'Poppins' font from Google Fonts and applies it to the entire document.
   * Font weights 300 (light), 400 (normal), and 700 (bold) are included.
2. **Global Styles:**
   * The universal selector (\*) and its pseudo-elements are reset to have no margin or padding, and the box-sizing property is set to "border-box" to include padding and borders in the element's total width and height.
3. **Body Styles:**
   * The body element has default styling, setting the font size, line height, text color, and background color.
4. **Header Styles:**
   * The header is styled as a flex container with items aligned to the center.
   * The h1 element within the header has a specific font weight and left margin.
5. **Container Styles:**
   * The container div is styled to have a maximum width, margin, and padding.
6. **Edit Mode Styles:**
   * The class **.edit-mode** is styled with a specific text color (#ccc).
7. **Form and Input Styles:**
   * Styles for general form inputs are defined, including width, font size, margin, padding, border, and border-radius.
   * There's a separate style for the filter input.
8. **Button Styles:**
   * Buttons have common styling, including background color, text color, border, border-radius, padding, and cursor properties.
   * Hover styles are specified for the primary button, changing the background color.
9. **Clear Button Styles:**
   * The "Clear All" button has specific styling, including margin, width, font size, background color, text color, border, border-radius, padding, and cursor properties.
   * Hover styles are defined for the clear button.
10. **Text-Red Class:**
    * The class **.text-red** is defined to set text color to red.
11. **Item List Styles:**
    * The list of items is styled as a flex container with a margin-top and flex-wrap properties.
    * List items have specific styling for display, justification, width, border, border-radius, padding, margin, and font weight.
    * A media query is used to adjust the width of list items when the viewport width is 500 pixels or less.

These styles collectively provide a clean and responsive layout for a shopping list application, with attention to typography, buttons, and the overall aesthetic. The media query ensures a more user-friendly experience on smaller screens.

**JavsScript**

1. **DOM Elements:**
   * Variables are declared to store references to various HTML elements using **document.getElementById**, **document.querySelector**, and other DOM manipulation methods.
   * Key elements include the form (**itemForm**), input field (**itemInput**), item list (**itemList**), clear button (**clearBtn**), filter input (**itemFilter**), and form button (**formBtn**).
2. **Event Handling:**
   * Event listeners are set up to respond to user interactions.
   * The **itemForm** is set to listen for the form submission event (**submit**).
   * The **itemList** is set to listen for click events (**click**).
   * The **clearBtn** is set to listen for click events (**click**).
   * The **itemFilter** is set to listen for input events (**input**).
   * The **DOMContentLoaded** event triggers the **displayItems** function when the page loads.
3. **Display Items:**
   * The **displayItems** function retrieves items from local storage using **getItemsFromStorage** and populates the item list by calling **addItemToDOM** for each item.
   * It also calls **checkUI** to update the display of certain elements based on the number of items.
4. **Add Item Functionality:**
   * The **onAddItemSubmit** function handles the form submission.
   * It prevents the default form behavior, trims the input value, and validates the input.
   * If in edit mode, it removes the existing item, updates the DOM, and clears edit mode.
   * Otherwise, it checks if the item already exists and adds it to the DOM and local storage.
5. **Remove Item Functionality:**
   * The **onClickItem** function is an event handler for item removal. It checks if the clicked element has the class **remove-item** and then removes the item from the DOM and local storage.
6. **Clear All Items Functionality:**
   * The **clearItems** function removes all items from the DOM and clears the items from local storage.
7. **Filter Items Functionality:**
   * The **filterItems** function filters items in real-time based on user input in the filter field.
8. **Local Storage Functionality:**
   * Functions like **getItemsFromStorage**, **addItemToStorage**, and **removeItemFromStorage** manage the interaction with local storage for persisting the shopping list.
9. **Edit Mode Functionality:**
   * The **setItemToEdit** function sets the edit mode, highlighting the selected item for editing and changing the form button text to "Update Item."
10. **UI Updates:**
    * The **checkUI** function updates the display of certain elements based on the number of items. It also resets the input field and manages edit mode.
11. **Initialization:**
    * The **init** function sets up event listeners on page load (**DOMContentLoaded**).
    * It also calls **checkUI** initially to update the UI based on the existing items in local storage.

Overall, this JavaScript code creates a dynamic and interactive shopping list application, managing item addition, removal, editing, and filtering, while persisting the data in local storage.

**What I learn**

1. **HTML:**
   * Document structure with **<html>**, **<head>**, **<body>**.
   * Use of various HTML elements like **<form>**, **<input>**, **<ul>**, **<li>**, **<button>**, **<header>**, and **<img>**.
2. **CSS:**
   * Styling using classes, IDs, and element selectors.
   * Use of flexbox for layout.
   * Responsive design with media queries.
   * Styling buttons, forms, and list items.
   * Importing external fonts (Google Fonts).
   * Creating and applying CSS classes for specific styles.
3. **JavaScript:**
   * Selecting DOM elements using **document.getElementById**, **document.querySelector**, etc.
   * Setting up event listeners for user interactions (e.g., form submission, button clicks, input events).
   * DOM manipulation methods such as **appendChild**, **removeChild**, and **querySelectorAll**.
   * Working with event objects to handle user actions.
   * Managing form submissions and preventing default behavior.
   * Conditional statements for validation and decision-making.
   * Utilizing functions to organize code and improve maintainability.
   * Real-time filtering of items based on user input.
   * Interacting with local storage to persist data (storing, retrieving, and removing items).
   * Dynamic creation of HTML elements using JavaScript.
   * Understanding and utilizing the concept of edit mode.
   * Utilizing the **localStorage** API for client-side data storage.
4. **Overall Project Structure:**
   * Proper separation of concerns (HTML, CSS, JavaScript files).
   * Linking external stylesheets and scripts.
   * Initializing the application with an **init** function.
   * Modularizing code into functions for specific tasks.
5. **Responsive Design:**
   * Adjusting the layout using media queries for smaller screens.
   * Making the application usable and visually appealing across different devices.
6. **UI/UX Considerations:**
   * User-friendly feedback through alerts.
   * Updating the display based on the number of items.
   * Clearing input fields after actions.
   * Highlighting items in edit mode.
   * Ensuring a smooth and intuitive user experience.
7. **External Dependencies:**
   * Linking external libraries or stylesheets (Font Awesome in this case) to enhance the visual presentation.
8. **Git/GitHub:**
   * Managing changes to the codebase, committing changes, and understanding version control concepts.

By working on this project, you've gained practical experience in building a dynamic web application that involves front-end technologies (HTML, CSS, JavaScript), interactivity, data persistence, and user interface considerations. This project covers a broad range of web development concepts and provides a solid foundation for further learning and exploration.