1. Script Initialization

This is the main wrapper that ensures the entire script runs only after the webpage's HTML content is fully loaded and ready.

- **document.addEventListener("DOMContentLoaded", async () => { ... });**: This is the entry point. The async keyword is used because the function contains await calls for loading data, which pauses execution until the data is fetched.

---

2. DOM Element Caching

At the beginning, the script finds all the necessary HTML elements it needs to interact with and stores them in constants for quick and easy access.

- **const container = document.getElementById(...)**: It gets references to the main content div (cards-container), the footer's year span, and the "Windows" and "Linux" buttons.

---

3. Data Loading from External JSON

This section is responsible for fetching the application's content from an external file, which keeps the data separate from the logic.

- **let data = {};**: An empty object is created to hold the card data once it's loaded.
- **async function loadCardData()**:
  - It uses the fetch('cards-data.json') API to make a network request for the JSON file.
  - await response.json() parses the text content of the file into a usable JavaScript object.
  - A **try...catch** block provides robust error handling. If the file is missing, or if it contains invalid JSON, an error is logged to the console and a user-friendly error message is displayed on the page.

---

4. UI Rendering

This function is responsible for dynamically creating the HTML for the cards based on the currently selected operating system.

- **function renderCards(os)**:
  - It first clears any existing content from the container to make way for the new cards.
  - It retrieves the correct array of sections (either data.windows or data.linux).
  - It loops through each section using forEach.
  - Inside the loop, it uses map() to transform each link object into an HTML list item (<li>) string.

- Finally, it assembles the full HTML for a card and appends it to the main container.

---

5. User Interaction and Animations

This function handles the logic when a user clicks on the "Windows" or "Linux" buttons.

- **function selectOS(os)**:
  - It first checks if the clicked button is already selected to prevent redundant animations.
  - **winBtn.classList.toggle(...)**: This efficiently adds or removes the .selected class to update the button's appearance.
  - **Animation Orchestration**:
    1. It adds the .fade-out CSS class to the container, triggering the fade-out animation.
    2. It uses addEventListener("animationend", ..., { once: true }) to wait for the animation to complete. This is more reliable than using a fixed setTimeout.
    3. Once the content is invisible, it calls renderCards(os) to swap the content.
    4. It then removes .fade-out and adds .fade-in to animate the new content into view.

---

6. Application Startup

These are the final lines of the script that kick everything off.

- **await loadCardData();**: The script pauses here to ensure the data is loaded before trying to display anything.
- **Event Listeners**: It attaches the click event listeners to the "Windows" and "Linux" buttons, linking them to the selectOS function.
- **Initial View**: It manually calls selectOS('windows') (or a similar initial setup) to ensure the page displays the "Windows" content by default when it first loads.