Website using HTML , CSS and JAVASCRIPT .

1. Document Setup (<head>)

The `<head>` section defines the document's metadata, links to external resources, and initial setup.

Tag/Element	Purpose	Details
`<!DOCTYPE html>`	Specifies the document type.	Declares the file as HTML5.
`<meta charset="UTF-8">`	Character encoding.	Ensures proper display of various characters.
`<meta name="viewport"...>`	Viewport configuration.	Essential for **responsive design**, ensuring the page scales correctly on different devices.
`<title> Visya </title>`	Browser tab title.	Displays " Visya " in the browser tab.
`<link rel="icon"...>`	Favicon.	Links to the site's icon in the browser tab.
`<link href="style.css"...>`	CSS Styling.	Links to the main stylesheet (`style.css`).
`<link rel="stylesheet"...>`	Additional CSS Styling.	Links to a secondary stylesheet (`style1.css`).

`<link rel="stylesheet"...>`	**Font Awesome Icons.**	Links to an external CDN for a comprehensive icon library (e.g., `fa-house`, `fa-burger-fries`).
`<script src="https://unpkg.com/sweetalert/dist/sweetalert.min.js">`	**SweetAlert Library.**	Includes a third-party library for creating customizable, attractive alert and dialog boxes, likely used for the booking confirmation.
`<script src="food.js"></script>`	**JavaScript Logic.**	Links to the main client-side JavaScript file that handles interactivity (e.g., menu filters, "Read More" functionality, booking logic).

HOME PAGE

Code html for first page :

This section covers the elements that create the main layout and navigation.

Element	Class/ID	Role/Purpose
`<div>`	`.hero-area`	**Main Container.** Wraps the header and hero content, serving as the background image section for the entire view height (`100vh`).
`<header>`	`.header-part`	**Navigation Wrapper.** Positions the navigation bar absolutely at the top of the page.
`<div>`	`.nav_bar`	**Navigation Layout.** Uses `flexbox` (`display: flex`) to align the logo and the navigation links horizontally, fixed to the top of the viewport (`position: fixed`).
`<span>`	`.logo`	**Restaurant Brand.** Displays the name "Visya" along with a food icon (`<i class="fa-light fa-burger-fries">`).
`<ul>`	`.nav_list`	**Link List.** Holds the main navigation items, styled to be inline.
`<a>`	(Multiple)	**Navigation Links.** Links to different pages (Home, Menu, About, Book Table). They include a Font Awesome icon (`<i>`) for visual flair.
`<a>`	`.nav_btn`	**Call-to-Action Button.** The "Order Online" link, styled distinctly as a button.
`<div>`	`.hero-content`	**Main Call-to-Action.** Contains the headline, descriptive paragraph, and the "Order Now" button. It's positioned and styled to sit above the image overlay.
`<h1>`	(None)	**Hero Headline.** Displays "Welcome to Foodie Hub."
`<a>`	`.btn`	**Action Button.** The "Order Now" link, directing to `booktable.html`.

CSS Styling and Design Features

This section details the primary CSS rules used to style the layout and provide visual effects.

Hero Area Styling

Selector	Property Used	Key Design Effect

`.hero-area`	`background,`
	`height,`
	`display: flex`
		Sets the **background image** (`img/bb img.jpg`), makes it cover the full screen height (`100vh`), and uses **Flexbox** to position the content to the right (`justify-content: flex-end`).

`.hero-area::before`	`content`, `position`, `background-color`	Creates a **dark overlay** (`rgba(24, 24, 24, 0.75)`) using a pseudo-element. This ensures the white text remains highly readable over the background image.

`.hero-content`	`position,`
	`z-index,`
	`max-width`,
	`margin-right`
		Ensures the text is **above the overlay** (`z-index:3`) and controls its size and position by floating it 150px from the right edge.

`.hero-content h1`	`font-family`, `color`	Uses the specific font **'Gravitas One'** and sets the accent color to a warm **orange/brown (`#C6702C`).**

1. "I established a **full-height hero section** with a striking background image and a **dark, semi-transparent overlay** to ensure maximum readability of the main text and calls to action."
2. "The hero content itself is **prominently positioned** and features the main value proposition, culminating in a clear **'Order Now' call-to-action button**."
3. "Crucially, the navigation bar is implemented as a **fixed element** with a smooth, CSS-only **sliding background hover effect** to enhance user engagement and visual branding.

Navigation Bar Styling

Selector	Property Used	Key Design Effect
`.nav_bar`	`position: fixed`, `background`, `display: flex`	Makes the navigation bar **sticky** at the top of the viewport. Uses Flexbox to space the logo and links apart (`justify-content: space-between`).
`.nav_bar .logo`	`font-family`, `color`, `margin-left`	Styles the logo text using **'Gravitas One'** and positions it with a margin.
`.nav_bar li a`	`color: black`, `transition`, `position: relative`	Sets the **default link color to black** (though it might be intended to be `antique white` like the logo or changed later) and prepares the link container for the hover animation by setting `position: relative`.
`.nav_bar li a::after`	`background`, `transform`, `transition`, `z-index:-2`	Creates a full-size **Saffron Gold background (`#C6702C`)** that is initially hidden by being moved off-screen to the right (`translate(100%)`). This is the key element for the **sliding hover effect.**
`.nav_bar li a:hover::after`	`transform: translateX(0)`	On hover, the gold background **slides fully into view** from the right, covering the link.
`.nav_bar li a:hover`, `.nav_bar li a:hover i`	`color`	Changes the text and icon color to **white** on hover, ensuring they contrast against the gold background.
`.nav_btn`	`background`, `padding`, `border-radius`	Styles the "Order Online" link as a distinctive **yellow/gold button (`#f1c40f`)** with rounded corners.

Menu Page

1. HTML (Structure and Content)

Selector/Element	Property/Attribute Used	Key Role/Effect
`<section class="menu">`	`style="margin-top: 80px;"`	Pushes the main menu content down to avoid overlap with a fixed navigation bar.
`<ul class="filters_menu">`	`data-filter="..."`	**Crucial for JavaScript Filtering:** Defines the category/class to filter menu items by (e.g., `.burger`).
`<div class="items">`	`class="... burger"` / `... pizza`	**Item Tagging:** Marks the item's category, which is matched by the `data-filter` value.
`<h3>` / `<p>` / `<h4>`	(Contained Text)	Stores the **Product Name**, **Description**, and **Price** data.
`<i class="... add-cart">`	`class="add-cart"`	**Action Trigger:** JavaScript listens for clicks on this class to initiate adding the product to the cart.
`<div class="cart">`	`class="cart"`	**Cart Container:** The parent element for the sliding sidebar.
`<div class="cart-content">`	(No unique attributes shown)	**Dynamic Insertion Point:** Where JavaScript adds all the selected cart item HTML.
`<button class="btn-buy">`	`onclick="btn()"`	**Checkout Trigger:** Calls the JavaScript function to start the order process.

2. CSS (Styling & Layout)

Selector	Property Used	Key Design Effect
`.menucontent`	`display: flex; justify-content: center; flex-direction: column;`	Centers the main title and filter bar vertically and horizontally.
`.filters_menu li`	`cursor: pointer; transition: 0.3s;`	Makes category names look clickable and ensures smooth color changes.
`.filters_menu li:hover, .active`	`background-color: #e65c00; border-radius: 10px;`	**Highlights** the active or hovered category button using the accent color.
`.items`	`flex: 1 1 300px;`	Creates a **responsive grid** layout where items maintain a minimum width and wrap as needed.
`.items .box:hover`	`transform: translateY(-8px); box-shadow: ...;`	Creates a modern **3D "lift-up" effect** when the cursor is over the product card.
`.img-box img:hover`	`transform: scale(1.2);`	Provides an **engaging zoom-in effect** on the product image.
`.items .box`	`background: linear-gradient(...)`	Gives the product card a subtle, appealing **gradient background**.
`.cart`	`position: fixed; right: -100%; transition: 0.3s; z-index: 100;`	Sets up the cart as a **fixed sidebar** that is initially hidden off-screen, ready to slide in smoothly.
`.cart.cart-active`	`right: 0;`	The state that forces the hidden cart to **slide into the visible area**.
`.cart-box`	`display: grid; grid-template-columns: 80px 1fr 30px;`	**Structured Cart Item:** Arranges the item image, details, and remove icon in three clean, vertical columns.
`.btn-buy`	`background-color: goldenrod;`	Defines the prominent, high-contrast style for the main **checkout button**.

3. JavaScript (Logic and Interactivity)

Selector/Function	Property/Method Used	Key Action/Logic
`btncart`, **`btnclose`**	`addEventListener("click", ...)`	**Cart Toggle:** Listens for clicks on the cart icon or close icon to show or hide the cart panel.
`cart`	`.classList.add("cart-active")` / `.remove(...)`	**Cart Visibility:** Adds or removes the `cart-active` class, causing the cart sidebar to slide in or out (based on CSS).
`attachAddToCartHandlers()`	`document.querySelectorAll(".add-cart")`	**Setup:** Finds all "Add to Cart" icons and attaches the `addCart` function as a click listener to each one.
`addCart(clickedEl)`	`clickedEl.closest(".items")`	**Product Detail Extraction:** Finds the parent food item (`.items`) to extract its title, price, and image source.
`addCart(clickedEl)`	`Array.from(...).includes(title)`	**Duplicate Check:** Ensures the item is not already in the cart by checking existing titles, preventing redundant additions.
`addCart(clickedEl)`	`document.createElement("div")`, `.innerHTML`	**Cart Item Creation:** Dynamically creates the full HTML structure for the new item (`cart-box`) and injects it into the `.cart-content`.

`removeItemHandler()`	`this.closest(".cart-box")?.remove()`	**Item Deletion:** Finds the parent cart item and removes it from the DOM, clearing it from the cart list.
`qtyChangeHandler()`	`isNaN(this.value)` / `this.value < 1`	**Input Validation:** Ensures the user's quantity input is a positive number (minimum 1).
`updateTotalAndCount()`	`parseFloat(priceElement.dataset.price)`	**Price Calculation:** Reads the base price from the `data-price` attribute and multiplies it by the quantity to get the line total.
`updateTotalAndCount()`	`totalEl.innerText = "Rs." + total.toFixed(2)`	**UI Update:** Calculates the final grand total, formats it to two decimal places, and updates the `.total-price` element.
`function btn()`	`document.querySelectorAll(".cart-box").length`	**Checkout Gate:** Checks if the cart contains any items before proceeding with the order.
`function btn()`	`cartContentEl.innerHTML = '';`	**Order Completion:** Clears the entire cart content and then calls `updateTotalAndCount()` to reset the total to zero.
`function btn()`	`swal(...)`	**Feedback:** Displays a stylized success message (simulating a receipt) after the order is "placed."

This JavaScript code is the **engine** behind your menu, handling all the interactive features like opening the cart, adding items, updating prices, and simulating the checkout process.

---

1. Initial Setup and Cart Visibility

This section links your JavaScript variables to the corresponding HTML elements and sets up the basic **open/close** functionality for the sidebar cart.

Code Section	Purpose	How it Works
`const btncart = ...`	**Element Selection**	Selects the HTML elements: the Cart Icon (`#cart-icon`), the Cart Panel (`.cart`), the Close Button (`#cart-close`), and the item counter badge (`.cart-count`).
Open Cart	**Event Listener**	When the **Cart Icon** is clicked, it adds the CSS class **`cart-active`** to the `.cart` element, which slides the cart panel into view (as defined by your CSS).
Close Cart	**Event Listener**	When the **Close Icon** is clicked, it removes the **`cart-active`** class, sliding the cart panel back out of view.
Initial Calls	**Setup**	Calls `attachAddToCartHandlers()` to make all existing menu items clickable and calls `updateTotalAndCount()` to initialize the cart total (which starts at 0).

---

2. Core Cart Handlers (Functions)

These functions manage the user interaction **within** the menu and the cart panel.

Function	Trigger	Action & Purpose

`attachAddToCartHandlers()`	Runs on page load and after certain updates.	Finds every **`.add-cart`** icon and attaches a click listener to it. It uses `btn.dataset.bound` to ensure the listener is only added once (preventing duplicate clicks).
`removeItemHandler()`	Click on the **Trash Icon** (`.cart-remove`).	1. Prompts the user for confirmation (`confirm`). 2. If confirmed, it uses `this.closest(".cart-box")` to find and remove the entire item from the cart list. 3. Immediately calls `updateTotalAndCount()`.
`qtyChangeHandler()`	Changing the value in the quantity input field.	1. **Validation:** Checks if the input is a valid number and ensures it is not less than 1. 2. Calls `updateTotalAndCount()` to reflect the new price.

3. `addCart()` Logic

This is the main function that handles getting product details and creating the dynamic HTML for the cart.

Step in `addCart()`	**Purpose**	**Key Technique**
Get Item Details	Extracts the item's title, price, and image URL from the clicked product card's parent container (`.items`).	Uses **DOM traversal** (`.closest()`, `.querySelector()`) to locate elements relative to the clicked icon.
Prevent Duplicates	Ensures the same item isn't listed multiple times.	Gets a list of titles already in the cart and shows an `alert` if the new item's title is already present.
Build Cart Box	Constructs the new item's HTML (`cartBox`).	Uses a **template literal** (the backtick string ``) to build the structure with the item's details (image, title, price, quantity input, remove icon).

Attach Handlers	Makes the new item interactive.	Explicitly attaches the `removeItemHandler` and `qtyChangeHandler` to the newly created trash icon and quantity input.

4. `updateTotalAndCount()` (The Calculator)

This is the central function responsible for all price and count updates.

Step in `updateTotalAndCount()`	Purpose	Calculation Details
Initialization	Sets the `total` and `count` variables to 0 before starting the calculation.	`let total = 0; let count = 0;`
Loop Through Items	Iterates over every single item currently visible in the cart (`.cart-box`).	Uses `cartBoxes.forEach(...)` to process each item one by one.
Calculate Line Total	Determines the total cost for the current item.	Multiplies the `unitPrice` (read from the `data-price` attribute for accuracy) by the item's `qty`.
Update Display	Updates the visible prices and counts on the page.	Updates the line total (`.cart-amt`), adds to the running `total`, updates the grand total (`.total-price`), and updates the item badge count (`.cart-count`).

5. `btn()` (Checkout/Payment Simulation)

This function runs when the user clicks the **"Place Order"** button.

Step in `btn()`	Purpose	Outcome

Empty Cart Check	Prevents ordering if nothing is in the cart.	Displays a warning message (using the styled **SweetAlert** function, `swal`).
Order Confirmation	Asks the user to confirm the purchase.	Uses a standard JavaScript `confirm` box showing the total price.
Clear Cart	Simulates a successful order completion.	Sets the entire content of `.cart-content` to empty (`innerHTML = ''`).
Final Update	Resets the UI total to zero.	Calls `updateTotalAndCount()` to zero out the total price and count badge.
Success Message	Alerts the user that the order is complete.	Displays a polished, custom success message using the **SweetAlert (`swal`)** library.

Explaining the "Dynamic" Menu Functionality

The term **"dynamic"** in your menu page refers to its ability to **change content and behavior without reloading the page**. This is achieved through the powerful synergy of HTML, CSS, and JavaScript.

The two main dynamic features are **Content Filtering** and the **Shopping Cart System**.

---

1. Dynamic Content Filtering (JS + HTML Classes)

This feature allows users to click a category (like "Pizza") and instantly see only the items belonging to that group.

Step	Technology	Action
1. User Action	HTML/JS	The user clicks a category button in the filter bar (e.g., `<li data-filter=".pizza">`).
2. Identify Target	JavaScript	The script reads the target selector from the button's **`data-filter`** attribute (e.g., `".pizza"`).

3. Content Manipulation	JavaScript	The script iterates through *all* menu item containers (`.items`).
4. Show/Hide	JavaScript/CSS	If an item's class matches the filter (e.g., it has the class `pizza`), the script sets its CSS property to **`display: flex`** (show). If it doesn't match, it sets it to **`display: none`** (hide).
Result	**Dynamic Display**	The list of menu items instantly changes to show only the selected category, creating a smooth, dynamic user experience without a page refresh.

2. Dynamic Shopping Cart (JS + HTML Generation)

This system dynamically manages the contents, calculations, and overall state of the cart sidebar.

Step	Technology	Action
1. Add Item	JavaScript	When you click an **`.add-cart`** icon, the `addCart()` function runs.
2. HTML Injection	JavaScript	The function **dynamically creates a new HTML structure** (a `<div class="cart-box">`) filled with the product's image, title, price, and quantity input.
3. Update Total	JavaScript	The `updateTotalAndCount()` function is called, which loops through **all** current `.cart-box` elements to recalculate the **grand total** and the **item count badge**.
4. Attach Events	JavaScript	Event listeners for **removing** the item (trash icon) and changing the **quantity** input are *dynamically attached* to the newly created HTML elements.
Result	**Real-time Updates**	The cart sidebar content and the total price update in **real-time** as the user adds, removes, or changes item quantities.

ABOUT US PAGE

This "About Us" section code demonstrates a common and effective web design pattern: the **Expandable Content Toggle** (Read More/Read Less).

1. HTML (Structure & Content)

Selector/Element	Property/Attribute Used	Key Role/Effect
`<div class="about">`	`class="about"`	Main container for the entire two-column section (Image + Text).
`<div class="a-img">`	`src="img/contentimg.png"`	Container for the visual element/image related to the "About" content.
`<div class="a-content">`	`class="a-content"`	Container for the textual content (Heading, Intro, and Hidden text).
`<p id="moreText">`	`style="display: none;"`	**Crucial:** Hides the extended "Read Less" content by default when the page loads.
`<a id="readMoreBtn">`	`id="readMoreBtn"`	**Action Trigger:** The element the JavaScript listens for clicks on to toggle the hidden text.
`<a class="btn">`	`class="btn"`	Provides the base styling for the call-to-action button.

---

2. CSS (Presentation and Styling)

Selector	Property Used	Key Design Effect

`.about`	`display: flex;`
	`align-items: center;`
	`justify-content: center;`
		Two-Column Layout: Arranges the image (`a-img`) and text (`a-content`) side-by-side and centers the entire block.

`.about`	`background-color: #e0e0e0;`	Gives the section a subtle, light grey background to set it apart from other content.
`.a-img img:hover`	`transform: scale(1.3);`	Creates a prominent **zoom-in effect** on the image when the cursor hovers over it, adding visual interest.

`.a-content h1`	`color: #e65c00;`
	`font-family: 'Gravitas One', ...;`
		Styles the heading with the **brand accent color** and a distinct font for emphasis.

`.a-content p`	`text-align: justify;`	Gives the body text a clean, newspaper-like look with straight left and right edges.

`.btn`	`background: #f1c40f;`
	`border-radius: 10px;`
		Styles the "Read More" link as a visually appealing, rounded button using a secondary brand color.

`.btn:hover`	`background: #e65c00;`
	`transition: 0.3s;`
		Provides **hover feedback** by changing the button color to the primary accent color with a smooth transition.

3. JavaScript (Logic and Interactivity)

Selector/Function	Property/Method Used	Key Action/Logic
`readMoreBtn`	`document.getElementById ("readMoreBtn")`	**Element Selection:** Finds the specific "Read More" button to attach the click listener.
`readMoreBtn`	`addEventListener ("click", function () { ... })`	**Toggle Trigger:** Listens for a click on the button to execute the expandable logic.
`moreText`	`moreText.style. display === "none"`	**State Check:** Determines the current visibility of the hidden content (`#moreText`).
`moreText`	`moreText.style. display = isHidden ? "block" : "none";`	**Visibility Toggle:** If the content is hidden, it sets it to `block` (shows it); if it's visible, it sets it back to `none` (hides it).
`this.textContent`	`this.textContent= isHidden ? "Read Less": "Read More";`	**Button Label Swap:** Changes the button's text from "Read More" to "Read Less" (and vice-versa) to reflect the new state.

Table Booking Page

Table Booking Page featuring a form for user input, client-side validation, and a backend email service integration (EmailJS) for confirmation.

1. HTML (Structure & Content)

Selector/Element	Property/Attribute Used	Key Role/Effect
`<div class="booking1">`	`style="margin-top: 80px;"`	Main container for the entire booking section, positioned below the header/nav bar.
`<iframe>`	`src="http://googleusercontent.com/..."`	**Display Map:** Embeds a placeholder for a Google Map location, showing users the physical location.
`<div class="book-table">`	`class="book-table"`	Container for the booking form itself, styled as a prominent box.
`<form class="details">`	`class="details"`	Container for all form fields; JavaScript uses this class to **reset** the form after submission.
`<input type="text" id="name">`	`id="name"`, `placeholder="..."`	Collects the user's name; linked to JS by its ID.
`<select id="myDropdown">`	`id="myDropdown"`	Collects the number of people; the **`selectedIndex`** property is used in JS for validation.
`<input type="date">`	`type="date"`	Provides a native date picker interface for the reservation date.

`<button type="button">`	`onclick="handleBooking()"`	**Action Trigger:** Calls the main JavaScript function (`handleBooking`) when clicked, instead of submitting the form traditionally.
`<div class="b-img">`	`src="img/back.png"`	Container for the large decorative side image.

---

2. CSS (Styling & Layout)

Selector	Property Used	Key Design Effect

`.booking1`	`display: flex;`
	`justify-content: center;`
	`align-items: center;`
		Full-Page Layout: Centers the map/image and the booking form box horizontally and vertically on the page.

`.booking1`	`background: url(...) no-repeat center center/cover;`	Sets a large **background image** for the entire section, creating a visual backdrop.

`.book-table`	`background: rgba(#888f7f);`
	`padding: 30px 40px;`
	`border-radius: 15px;`
		Styles the form as a clean, rounded, slightly transparent box for high visibility.

`.book-table h2`	`color: #f1c40f;`
	`font-family: Gravitas One;`
		Styles the main heading using the secondary **brand accent color** and a distinct font.

`.details input:focus, .details select:focus`	`border-color: #ff6600;`
	`box-shadow: 0 0 5px rgba(255, 102, 0, 0.9);`
		Provides crucial **visual feedback** (highlighting) when a user clicks into an input field.

`button`	`background: #f1c40f;`
	`transition: 0.3s;`
		Styles the "Book Now" button using the accent color and adds a smooth transition for hover.

`button:hover`	`background: #e65c00;`
	`color: white;`
		Change the button color to the **primary accent color** on hover.

`.b-img img`	`height: 100%;`
	`object-fit: cover;`
		Ensures the decorative side image stretches vertically to fill the height of its container.

---

3. JavaScript (Logic and Interactivity)

Selector/Function	Property/Method Used	Key Action/Logic	Working/Flow
`handleBooking()`	`document.getElementById('...').value.trim()`	**Input Collection:** Gathers data from all form fields (name, email, phone, etc.).	**Start:** Triggered when the user clicks the "Book Now" button.
`handleBooking()`	`!name || !email || !phone...`	**Required Field Validation:** Checks if any necessary field is empty.	**Step 2:** If any field is empty, an `alert` is shown, and the function stops (`return`).
`handleBooking()`	`emailPattern.test(email)`/`phonePattern.test(phone)`	**Format Validation:** Uses **Regular Expressions** (`/^[^\s@]+@[^\s@]+\.[^\s@]+$/`) to ensure email and phone follow required patterns.	**Step 3:** If formats are invalid, an `alert` is shown, and the function stops.

`handleBooking()`	`sendemail(name, email, ...)`	**Delegation:** Calls the dedicated function to communicate with the external email service.	**Final Step (Success):** Passes validated data to the next function.
`sendemail()`	`emailjs.send(SERVICE_ID, TEMPLATE_ID, templateParams)`	**API Integration:** Uses the **EmailJS library** to send the collected form data to the predefined email template.	**External Call:** Transmits data to the EmailJS server.
`sendemail()`	`.then(function(response) { ... })`	**Success Handling:** Executes upon successful email delivery. Displays a **SweetAlert** confirmation and clears the form using `document.querySelector('.details').reset()`.	**Result (Success):** Confirms booking and resets UI.
`sendemail()`	`.catch(function(error) { ... })`	**Error Handling:** Executes if the email transmission fails. Displays an error SweetAlert.	**Result (Failure):** Alerts the user that the booking attempt failed.

Footer page

This code creates a standard **three-column footer** layout followed by a copyright section. It uses Flexbox to structure the main content and provides contact information, branding, social links, and opening hours.

1. HTML (Structure & Content)

Selector/Element	Property/Attribute Used	Key Role/Effect
`<footer>`	(No unique attribute shown)	The semantic HTML wrapper, marking the content as the main page footer.
`<div class="footers">`	`class="footers"`	The main container for all footer content, applying the background color and padding.
`<div class="one">`	`class="one"`	**Three-Column Container:** Holds the 'left', 'middle', and 'right' sections, designed for Flexbox alignment.
`<div class="left">`	`<i class="fa-solid fa-...">`	The **Contact** column, displaying icons for location, phone, and email using Font Awesome.
`<div class="middle">`	`<h3>VISYA</h3>`	The **Branding** column, featuring the restaurant name and a short tagline.
`<ul class="social">`	`class="social"`	**Social Links:** A Flex container for the social media icons (Facebook, Instagram, YouTube).
`<div class="right">`	(Contained Text)	The **Hours** column, listing the daily operating schedule.
`<div class="two">`	`class="two"`	The bottom section is reserved for **Copyright** and template attribution.

---

2. CSS (Styling & Layout)

Selector	Property Used	Key Design Effect
`.footers`	`background: #888f7f; color: black;`	Sets the entire footer area to a **solid background color** (grey/taupe) with black text.
`.footers .one`	`display: flex; justify-content: space-around;`	**Three-Column Flexbox:** Distributes the 'left', 'middle', and 'right' columns evenly across the footer width.
`.footers .two`	`padding-top: 100px; text-align: center;`	**Copyright Section:** Pushes the copyright text down for vertical spacing and centers the text.
`.footers div h3`	`font-family: Gravitas One; font-size: medium;`	Styles the **section headings** ("contact us," "VISYA," "opening Hours") with a specific font and slightly smaller size.
`ul`	`display: flex; list-style: none;`	Converts the unordered list (used for social icons) into a **horizontal row**.
`.social`	`gap: 15px; padding: 10px 0px 0px 120px;`	**Social Spacing:** Adds space between the icons (`gap`) and pushes the entire icon group slightly to the right with padding.
`.footers h3`	`font-size: 20px; font-family: cursive;`	This selector is likely for the main heading (if one existed outside the columns), styling it with a larger size and cursive font.

Technical Structure Overview

HTML (Structure)

The project uses clean, semantic HTML to define distinct sections:

- **Menu Items:** Heavily relies on **class names** (e.g., `.burger`, `.pizza`) and **IDs/attributes** (`#cart-icon`, `data-filter`) as anchors for JavaScript.
- **Forms:** Uses standard form elements (`<input>`, `<select>`) within a `<form>` container, with the submission handled by a JS function (`onclick="handleBooking()"`) rather than a traditional form submit.

CSS (Design & Layout)

CSS is critical for the project's polish and responsiveness:

- **Flexbox Layouts:** Used extensively to create the three-column footer, center content on the booking page, and manage the responsive grid for menu items.
- **Visual Feedback:** Implements smooth transitions and transform effects (e.g., card lift-up, image zoom, button color change) to enhance user engagement.
- **Sliding Elements:** Uses **`position: fixed`** and CSS classes (`.cart-active`) to control the smooth appearance and disappearance of the shopping cart sidebar.

JavaScript (Interactivity)

The JavaScript is the core **application logic**, managing all dynamic behavior:

- **Event Handling:** Attaches click, change, and load listeners to trigger all primary actions (filter, add to cart, change quantity, place order).
- **DOM Manipulation:** Dynamically **creates** the HTML for new cart items and **modifies** existing element styles (e.g., changing `display: none` to `display: block`).
- **External Integration:** Utilizes the **EmailJS library** to connect the front-end booking form directly to an email service for sending notifications.
- **Validation:** Contains robust checks for field emptiness and the format of email/phone numbers to ensure reliable data submission.