ZOLAXTECH RESEARCH HUB

HTML, CSS & JavaScript

The Complete Beginner's Guide to Building a Website

Everything you need, in one document, to go from a blank text file to a complete, working, responsive website — no prior coding experience required.

What's inside

- Part One - HTML: page structure, text, links, images, lists, forms, and semantic layout
- Part Two - CSS: selectors, the box model, colors and typography, Flexbox, Grid, and responsive design
- Part Three - JavaScript: variables, functions, the DOM, events, and form validation
- Part Four - Capstone Project: one complete real website, built file by file, with screenshots of the finished result

Prepared by **ZolaXTech** · Free for students and beginners

Table of Contents

(Right-click this field in Word and choose "Update Field" once opened, so the page numbers fill in.)

How to Use This Guide?

This single document takes you from a completely blank page to one full, real, working website. It's organized in four parts, and each one builds on the last:

- Part One teaches HTML - the structure of a page (what content IS).
- Part Two teaches CSS - the style of a page (what content LOOKS like).
- Part Three teaches JavaScript - the behavior of a page (what content DOES).
- Part Four combines all three into one real, complete project: a small business website called Aurora Café, built file by file, ending with screenshots of the finished, working result.

> ✨**You need nothing but a plain text editor (Notepad, TextEdit, or ideally the free Visual Studio Code) and a web browser you already have. No installation, sign-up, or internet connection is required to build and view every example in this guide.**

A simple way to think about the three languages together:

Language	Role
HTML	The skeleton — headings, paragraphs, images, forms: what exists on the page
CSS	The skin and clothing — colors, fonts, spacing, layout: how it looks
JavaScript	The muscles and reflexes — clicks, menus, validation: how it behaves

PART ONE

HTML - The Structure

Building the skeleton every web page is made of

1.1 The Page Skeleton

Every HTML page starts with the exact same skeleton. Save this as index.html and it's a complete, valid web page:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>My First Page</title>
  </head>
  <body>
```

```
    <h1>Hello, World!</h1>
  </body>
</html>
```

Tag	What It Does
<!DOCTYPE html>	Tells the browser this is a modern HTML5 page. Always the first line.
<html>	Wraps the entire page.
<head>	Page information not shown directly: title, linked CSS/JS files, meta tags.
<meta viewport ...>	Tells mobile browsers to size the page to the device screen — essential for responsive design (Part Two).
<title>	Text shown on the browser tab.
<body>	Everything visitors actually see.

⚠**Common Mistake:** Forgetting to close a tag is the #1 beginner error. Every opening tag needs a matching closing tag, closed in reverse order.

1.2 Text: Headings and Paragraphs

```
<h1>Main Page Title</h1>
<h2>A Section Heading</h2>
<p>A regular paragraph of text.</p>
<strong>Bold, important text</strong>
<em>Italic, emphasized text</em>
```

Use exactly one <h1> per page (the main title), then <h2>, <h3>, etc. for sub-sections in order — never skip a level just to change size (that's CSS's job).

1.3 Links and Images

```
<a href="https://example.com">Visit a website</a>
<a href="about.html">Go to another page on my site</a>
<img src="photo.jpg" alt="A description of the photo">
```

href and src and alt are attributes — extra information inside the opening tag, always written as name="value". Never skip alt on an image: it's read aloud by screen readers and shown if the image fails to load.

1.4 Lists and Tables

```
<ul>
  <li>Unordered item</li>
</ul>
<ol>
  <li>Ordered, numbered item</li>
</ol>
<table>
  <tr><th>Header</th></tr>
  <tr><td>Data cell</td></tr>
</table>
```

1.5 Organizing a Page: Semantic Layout Tags

These tags describe the role of each section, making a page easier to read, style, and search-index than generic s alone:

Tag	Typical Use
<header>	Introductory area — logo, site title, main nav.
<nav>	A group of navigation links.
<main>	The primary content of the page (used once).
<section>	A distinct themed grouping of content.
<footer>	Closing info — copyright, contact links.
/	Generic containers with no built-in meaning — used for layout and styling hooks.

1.6 Forms: Collecting Input

```
<form>
  <label for="name">Name:</label>
  <input type="text" id="name" name="name">
  <textarea id="message"></textarea>
  <button type="submit">Send</button>
</form>
```

The for attribute on <label> should match the id on its <input> — clicking the label then focuses the field automatically. In Part Three, you'll use JavaScript to validate this form before it submits.

1.7 Comments and Best Practices

```
<!-- Comments like this are ignored by the browser -->
```

- Indent nested tags consistently (2 spaces is common).
- Always close tags, in the right order (last opened, first closed).
- Use lowercase tag and attribute names.
- Save your homepage as index.html — servers look for this name by default.

> ✨**This is a condensed refresher. For a deeper walkthrough of every HTML tag with individual examples, see ZolaXTech's dedicated "HTML for Beginners" guide.**

PART TWO

CSS - The Style

Turning a plain HTML skeleton into a designed page

2.1 What Is CSS, and How Do You Add It?

CSS (Cascading Style Sheets) controls how HTML looks: colors, fonts, spacing, and layout. There are three ways to add it — always prefer the third:

1. Inline (avoid — hard to maintain)

```
<p style="color: red;">Red text</p>
```

2. Internal (fine for a single quick page)

```
<head>
  <style>
    p { color: red; }
  </style>
</head>
```

3. External (recommended for every real project)

```
In your HTML <head>:
<link rel="stylesheet" href="style.css">
```

Then create a separate file named style.css in the same folder. This keeps structure (HTML) and style (CSS) cleanly separated — the approach used throughout the rest of this guide.

2.2 Selectors: Targeting What to Style

```
p { color: navy; }  /* every <p> */
.highlight { background: yellow; }  /* class="highlight" */
```

```
#main-title { font-size: 32px; }  /* id="main-title" */
nav a { text-decoration: none; }  /* <a> inside <nav> */
h1, h2, h3 { font-family: Georgia, serif; }
```

Selector	Matches
p	Every element of that tag type
.classname	Every element with class="classname" (reusable, most common)
#idname	The one element with id="idname" (must be unique per page)
parent child	Elements matching "child" nested inside "parent"
a:hover	An <a> element while the mouse is hovering over it

✨**Use classes for anything you'll style more than once, and reserve ids for one-off elements you also need to target with JavaScript (Part Three).**

2.3 The Box Model

Every HTML element is a rectangular box made of four layers, from the inside out: content, padding, border, and margin.

```
.card {
  width: 260px;        /* content width */
  padding: 20px;     /* space inside the border */
  border: 1px solid #ccc;
  margin: 16px;      /* space outside the border */
}
```

Layer	Purpose
content	The actual text/image, sized by width and height
padding	Breathing room between the content and its border
border	A visible (or invisible) line around the padding
margin	Space between this box and its neighbors

⚠**Common Mistake:** By default, width and height only cover the content — adding padding/border makes the box bigger than you set. Add box-sizing: border-box; to * at the top of your stylesheet, and width/height will include padding and border instead — far more predictable.

```
* { box-sizing: border-box; }  /* put this at the very top of every stylesheet */
```

2.4 Colors, Backgrounds, and Typography

```
body {
  color: #2E2A26;             /* text color */
  background-color: #F4EDE4;
  font-family: 'Segoe UI', Arial, sans-serif;
  font-size: 16px;
  line-height: 1.6;           /* space between lines of text */
}
h1 { font-weight: bold; text-align: center; }
```

Colors can be named (red), hex codes (#C98A3E), or rgb() values (rgb(201, 138, 62)) — hex codes are the most common in real projects.

2.5 Display and Flexbox: Real Layout

The display property decides how an element behaves in the page flow. Flexbox (display: flex) is the modern, reliable way to arrange elements in a row or column — used for nearly every navigation bar, card row, and button group on the modern web.

```
.card-grid {
  display: flex;
  flex-wrap: wrap;        /* let items drop to a new line if needed */
  justify-content: center;  /* horizontal alignment */
  gap: 24px;              /* space between items — no margin needed */
}
```

Property	Controls
display: flex	Turns on flexbox for this element's direct children
flex-direction	row (default, side by side) or column (stacked)
justify-content	Alignment along the main axis: flex-start, center, space-between
align-items	Alignment along the cross axis: flex-start, center, stretch
flex-wrap	wrap lets items drop to a new row instead of squeezing
gap	Even spacing between items, without extra margin rules

2.6 A Quick Look at CSS Grid

CSS Grid is Flexbox's sibling — better suited when you need rows AND columns to line up together, like a photo gallery or dashboard.

```
.gallery {
  display: grid;
```

```
  grid-template-columns: repeat(3, 1fr);  /* 3 equal columns */
  gap: 16px;
}
```

✨**Rule of thumb: reach for Flexbox when arranging items in a single row or column (like a nav bar or a row of cards); reach for Grid when you need a true two-dimensional layout (like a photo gallery).**

2.7 Positioning

Value	Behavior
static (default)	Normal page flow; top/left have no effect
relative	Normal flow, but top/left/right/bottom nudge it from where it would normally sit
absolute	Removed from normal flow; positioned relative to its nearest positioned ancestor
fixed	Removed from flow; stays in place even when the page scrolls (e.g., a sticky header)
sticky	Acts normal until a scroll threshold, then sticks in place — used for the nav bar in Part Four

```
.site-header {
  position: sticky;
  top: 0;
}
```

2.8 Responsive Design: Media Queries

A media query applies a block of CSS only when a condition is true — most commonly, only below a certain screen width — allowing your layout to adapt from desktop to mobile.

```
@media (max-width: 700px) {
  .card-grid {
    flex-direction: column;  /* stack cards on small screens */
  }
}
```

⚠**Common Mistake:** Without the <meta name="viewport" ...> tag from section 1.1 in your HTML <head>, media queries won't work correctly on real phones — mobile browsers will render the page at desktop width and then shrink it.

2.9 Common UI Patterns: Buttons and Cards

```
.btn {
  display: inline-block;
  background: #C98A3E;
  color: white;
  padding: 12px 28px;
```

```
  border-radius: 6px;
  text-decoration: none;
}
.btn:hover { background: #B5762E; }
```

You'll see this exact button and card pattern reused in the full capstone project in Part Four.

PART THREE

JavaScript - The Behavior

Making your page respond, react, and interact

3.1 What Is JavaScript, and How Do You Add It?

JavaScript (JS) is the programming language that runs inside the browser, letting a page react to clicks, validate a form, or update content without reloading. Like CSS, it can be inline, internal, or external — always prefer external:

```
Just before the closing </body> tag:
<script src="script.js"></script>
```

✨**Place your <script> tag at the end of <body>, not in <head>. This way, the browser finishes building the page's HTML elements first, so your JavaScript can safely find and use them.**

3.2 Variables and Data Types

```
let name = "Selam";        // text (a "string") — can change later
const age = 21;            // a number — const means it won't be reassigned
let isStudent = true;      // true or false (a "boolean")
let price = 49.99;
```

Keyword	When to Use
let	A variable whose value may change later
const	A variable that should never be reassigned (preferred by default)
var	Older syntax — avoid in new code; let/const replaced it

3.3 Operators and Conditionals

```
let score = 85;

if (score >= 90) {
```

```
  console.log("Grade: A");
} else if (score >= 80) {
  console.log("Grade: B");
} else {
  console.log("Grade: C or below");
}
```

Operator	Meaning
=== / !==	Equal to / not equal to (always use these, not = or ==)
> < >= <=	Greater than, less than, and their "or equal to" versions
&& / ||	Logical AND / OR, for combining conditions
+ - * /	Addition, subtraction, multiplication, division

3.4 Functions

A function is a reusable, named block of code. You define it once, then call it as many times as you need.

```
function greet(name) {
  return "Hello, " + name + "!";
}

console.log(greet("Selam"));  // "Hello, Selam!"
```

Modern JavaScript also uses a shorter "arrow function" syntax, especially for short functions passed to other code (you'll see this with events in section 3.9):

```
const greet = (name) => "Hello, " + name + "!";
```

3.5 Arrays and Objects

```
const drinks = [
  "Latte","Cold Brew","Espresso"
];
console.log(drinks[0]);        // "Latte" — arrays start counting at 0
console.log(drinks.length);     // 3

const product = {
  name: "Signature Latte",
  price: 120,
  inStock: true
};
console.log(product.name);       // "Signature Latte"
```

Arrays hold ordered lists of items (accessed by number, starting at 0). Objects hold named properties (accessed by name) — you'll use both constantly once you start working with real data.

3.6 Loops

```
for (let i = 0; i < drinks.length; i++) {
  console.log(drinks[i]);
}
// A cleaner modern alternative for arrays:
drinks.forEach((drink) => {
  console.log(drink);
});
```

3.7 The DOM: Selecting Elements

The DOM (Document Object Model) is JavaScript's live map of every HTML element on the page. Before you can change or react to something, you must first select it:

```
document.getElementById("contactForm");      // one element, by its id
document.querySelector(".btn");             // the first match for any CSS selector
document.querySelectorAll(".card");         // ALL matches, as a list
```

✨ **querySelector and querySelectorAll accept any valid CSS selector — the exact same selectors you learned in Part Two (.class, #id, tag names) — which is why learning CSS selectors first pays off immediately here.**

3.8 Changing Content and Styles

```
const heading = document.querySelector("h1");
heading.textContent = "Welcome Back!";       // change the text
heading.style.color = "#C98A3E";             // change a CSS style directly
heading.classList.add("highlight");         // add a CSS class instead (preferred)
heading.classList.toggle("open");           // add it if missing, remove if present
```

Prefer classList.add / toggle over setting .style directly — it keeps your visual rules in CSS where they belong, and JavaScript just flips a switch (exactly how the mobile menu works in Part Four).

3.9 Events: Reacting to the User

```
const button = document.querySelector(".btn");

button.addEventListener("click", () => {
  alert("Thanks for clicking!");
});
```

Event	Fires When
click	The user clicks an element
submit	A <form> is submitted (combine with event.preventDefault() to stop the page reloading)
input	The value of a text field changes, as the user types
DOMContentLoaded	The HTML has finished loading — useful for setup code (on document, not an element)

3.10 Putting It Together: Form Validation

Here's a realistic, complete pattern — checking a contact form isn't empty before letting it submit — combining almost everything from Part Three:

```
const form = document.getElementById("contactForm");

form.addEventListener("submit", (event) => {
  event.preventDefault();  // stop the page from reloading

  const name = document.getElementById("name").value.trim();

  if (name === "") {
    alert("Please enter your name.");
    return;
  }

  alert("Thanks, " + name + "!");
  form.reset();
});
```

This exact pattern — extended slightly — is what powers the contact form in the full capstone project you'll build next.

PART FOUR

Capstone Project

One real, complete website — Aurora Café — built with everything above

4.1 What You're Building

To bring HTML, CSS, and JavaScript together, this final part builds one complete, real website from scratch: a one-page site for a fictional coffee shop, Aurora Café. It includes a sticky navigation bar with a mobile menu, a

hero section, an about section, a row of menu cards, and a validated contact form — every technique from Parts One through Three, used for a real purpose.

You will create three files in the same folder: index.html, style.css, and script.js — exactly the external-file structure recommended in Parts Two and Three.

> ✨**Create a new folder on your computer called aurora-cafe, and create the three empty files inside it now. You'll fill each one in as you follow along.**

4.2 Step 1: The HTML Structure (index.html)

This is the full markup — a header with navigation, then four content sections, then a footer. Notice it links both style.css and script.js, exactly as taught in Parts Two and Three.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Aurora Café</title>
  <link rel="stylesheet" href="style.css">
</head>
<body>

  <header class="site-header">
    <div class="container nav-wrap">
      <a href="#home" class="logo">Aurora Café</a>
      <button id="navToggle">&#9776;</button>
      <nav id="siteNav">
        <a href="#home">Home</a>
        <a href="#about">About</a>
        <a href="#menu">Menu</a>
        <a href="#contact">Contact</a>
      </nav>
    </div>
  </header>
```

The nav toggle button and menu links will come alive with JavaScript in Step 3 — for now, they're just structure.

The Hero Section

```
  <section id="home" class="hero">
    <div class="container">
      <h1>Fresh Coffee, Warm Moments</h1>
      <p>Handcrafted drinks and honest food in the heart of the city.</p>
      <a href="#menu" class="btn">See Our Menu</a>
    </div>
  </section>
```

The About and Menu Sections

```
  <section id="about" class="about">
    <div class="container">
      <h2>About Aurora</h2>
      <p>Aurora Café opened in 2021 with one simple goal ...</p>
    </div>
  </section>

  <section id="menu" class="menu">
    <div class="container">
      <h2>Our Favorites</h2>
      <div class="card-grid">
        <div class="card">
          <h3>Signature Latte</h3>
          <p>Double espresso, steamed milk, a hint of vanilla.</p>
          <span class="price">120 ETB</span>
        </div>
        <!-- two more .card divs follow the same pattern -->
      </div>
    </div>
  </section>
```

✨**Notice card-grid is styled with display: flex in the CSS below (section 4.3) — this repeated .card pattern is exactly the Flexbox layout from section 2.5.**

The Contact Section and Footer

```
  <section id="contact" class="contact">
    <div class="container">
      <h2>Visit or Message Us</h2>
      <form id="contactForm">
        <label for="name">Name</label>
        <input type="text" id="name">
        <label for="email">Email</label>
        <input type="email" id="email">
        <label for="message">Message</label>
        <textarea id="message"></textarea>
        <button type="submit" class="btn">Send Message</button>
        <p id="formStatus"></p>
      </form>
    </div>
  </section>

  <footer class="site-footer">
    <p>&copy; 2026 Aurora Café.</p>
  </footer>

  <script src="script.js"></script>
</body>
</html>
```

⚠**Common Mistake:** The <script> tag is the very last thing before </body>, exactly as recommended in section 3.1 — this guarantees every element above it already exists when script.js runs.

4.3 Step 2: The Stylesheet (style.css)

Now the visual design. This is organized top-to-bottom exactly as the page reads — reset, header/nav, hero, about, menu cards, contact form, footer, and finally the mobile media query.

```
/* Reset & base */
* { box-sizing: border-box; margin: 0; padding: 0; }
body { font-family: 'Segoe UI', Arial, sans-serif; line-height: 1.6; }
.container { max-width: 1000px; margin: 0 auto; padding: 0 24px; }
```

```
/* Sticky header + flexbox nav */
.site-header { background: #2E2A26; position: sticky; top: 0; }
.nav-wrap {
  display: flex;
  justify-content: space-between;
  align-items: center;
}
.site-nav a { color: #F4EDE4; margin-left: 28px; }
.nav-toggle { display: none; }  /* shown only on mobile — see media query below */
```

```
/* Hero section */
.hero {
  background: linear-gradient(rgba(46,42,38,0.55), rgba(46,42,38,0.55)), #C98A3E;
  color: white;
  text-align: center;
  padding: 110px 0;
}
.btn {
  display: inline-block; background: #C98A3E; color: white;
  padding: 12px 28px; border-radius: 6px; text-decoration: none;
}
```

```
/* Menu cards — Flexbox in action */
.card-grid { display: flex; flex-wrap: wrap; justify-content: center; gap: 24px; }
.card {
  background: white; border-radius: 10px; padding: 28px 22px; width: 260px;
  box-shadow: 0 4px 14px rgba(0,0,0,0.08);
}
```

```
/* Contact form */
.contact form { max-width: 480px; margin: 0 auto; }
.contact input, .contact textarea {
  width: 100%; padding: 10px 12px; border: 1px solid #D8D2C8; border-radius: 6px;
}
```

```
/* Responsive: collapse nav into a mobile menu below 700px */
@media (max-width: 700px) {
  .nav-toggle { display: block; }
  .site-nav { display: none; flex-direction: column; width: 100%; }
  .site-nav.open { display: flex; }  /* JavaScript adds this class — see Step 3 */
}
```

✨**This media query is exactly the pattern from section 2.8. Below 700px, the full nav hides and the hamburger button appears — but only JavaScript can actually toggle it open. That's Step 3.**

4.4 Step 3: The Interactivity (script.js)

Two independent features, both using the DOM-selection, event, and class-toggling patterns from Part Three.

Feature 1: Mobile Menu Toggle

```
const navToggle = document.getElementById("navToggle");
const siteNav = document.getElementById("siteNav");

navToggle.addEventListener("click", function() {
  siteNav.classList.toggle("open");
});
```

This is section 3.8's classList.toggle in action: clicking the hamburger button adds the .open class defined in the CSS media query above, sliding the menu into view; clicking again removes it.

Feature 2: Contact Form Validation

```
const contactForm = document.getElementById("contactForm");
const formStatus = document.getElementById("formStatus");

contactForm.addEventListener("submit", function(event) {
  event.preventDefault();

  const name = document.getElementById("name").value.trim();
  const email = document.getElementById("email").value.trim();
  const message = document.getElementById("message").value.trim();

  if (name === "" || email === "" || message === "") {
    formStatus.textContent = "Please fill in every field before sending.";
    return;
  }

  formStatus.textContent = "Thanks, " + name + "! Your message has been sent.";
  contactForm.reset();
});
```

This is exactly the pattern built up across sections 3.9 and 3.10: select the elements, listen for the submit event, prevent the default page reload, check the values, and give the user clear feedback.

4.5 The Result: Your Finished Website

With all three files saved in the same folder, double-click index.html — this is what opens (shown here in a desktop browser, with a placeholder amber background standing in for a hero photo):

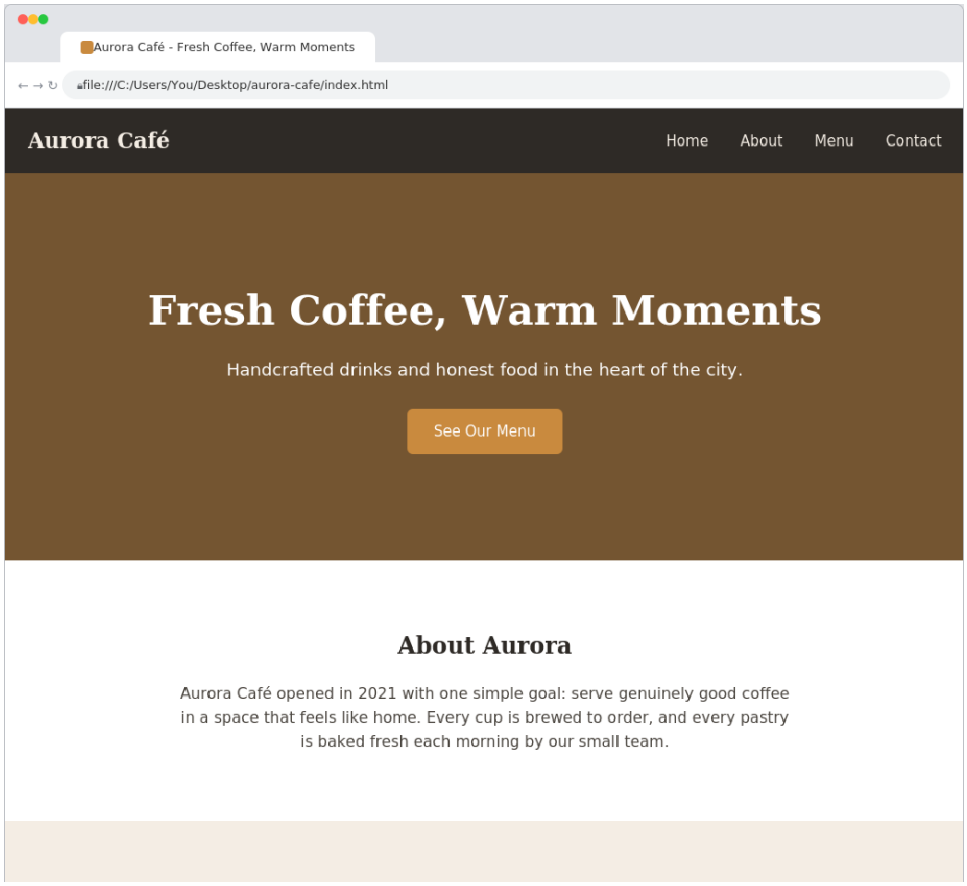

Figure 4.1: Aurora Café - home, hero, and about section

Scrolling down reveals the menu cards, arranged in a responsive row using the Flexbox layout from section 2.5 and 4.3:

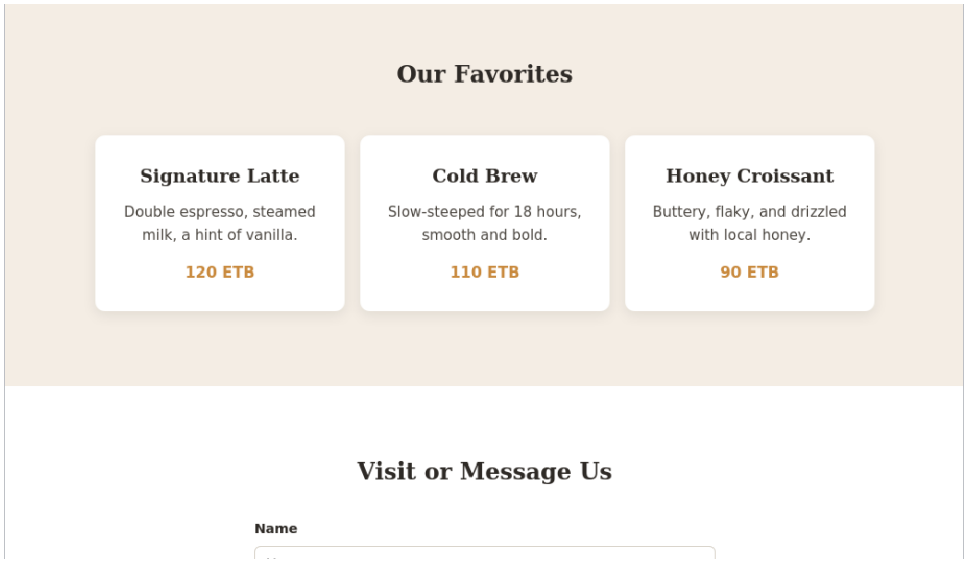

Figure 4.2: The menu section — three .card elements inside one flex .card-grid

And on a narrow screen, the responsive media query and JavaScript toggle work together: the nav bar collapses to a hamburger icon, ready to expand into a full menu on tap:

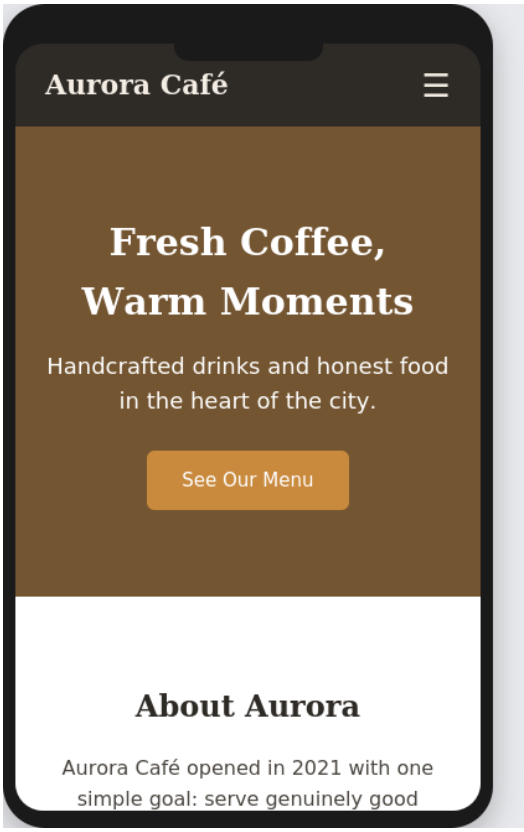

Figure 4.3: The same page at mobile width, with the collapsed navigation menu

4.6 What to Try Next

- Replace the hero background and profile-style content with your own business, portfolio, or personal idea.
- Add a fourth menu card, or a new section entirely, reusing the. card-grid pattern.
- Try changing the color values in style.css to see how quickly a full re-theme is possible once structure and style are separated.
- Once comfortable, look up free hosting options (e.g., GitHub Pages, Netlify) to publish your page live on the internet, visible to anyone with the link.

> ✨**You now have a complete, working three-file project structure — index.html, style.css, script.js — that you can copy as the starting point for your very next website.**

This guide covers the essentials needed to build and style a complete, responsive, interactive website. For the full depth of any topic, the Mozilla Developer Network (developer.mozilla.org) is the most reliable free reference to consult next.

Prepared by **ZolaXTech** — free to download, print, and share with beginners.