**AMAL-NAMA** is actually built using a very standard, logical pattern called a **Single Page Application (SPA)**.

### The Big Picture Analogy

1. **HTML (index.html) is the Building:** It creates the walls, the tables, the kitchen, and the menu. It defines *structure*, but it's empty and unpainted.
2. **CSS (styles.css) is the Interior Design:** It paints the walls purple, makes the text look nice, ensures the tables are arranged in a grid, and decides what is visible or hidden.
3. **JavaScript (app.js, data.js, utils.js) is the Staff:**
   * **data.js** is the **Pantry/Inventory Manager**. It remembers who the users are, what grades they have, and keeps the records safe.
   * **utils.js** is the **Specialist Toolset**. Need to print a receipt (PDF)? Need to announce an order (Toast notification)? This file handles specific, reusable tasks.
   * **app.js** is the **General Manager**. It watches the customers (you), listens for clicks, tells the Inventory Manager to get data, and tells the building (HTML) to update what is shown on the walls.

### Phase 1: The Structure (HTML & CSS)

1. The One-Page Trick (index.html)

This app doesn't actually have different "pages" like login.html or dashboard.html. It has one file: index.html.

Look at this part of the HTML:

HTML

<div id="app">  
 <div id="login-page" class="page active"> ... </div>  
 <div id="dashboard-page" class="page"> ... </div>  
</div>

* **Mechanism:** Both the Login screen and the Dashboard exist at the same time.
* **The Trick:** CSS hides everything by default. The class active makes *only* that specific section visible. When you "switch pages," JavaScript simply removes the active class from the Login div and adds it to the Dashboard div.

**2. The Styling (styles.css)**

* **Variables:** At the top, you see :root { --primary-900: #1e1033; ... }. This defines a palette. Instead of typing #1e1033 everywhere, the developer types var(--primary-900). If they want to change the theme from Purple to Blue, they change it here once, and the whole app updates.
* **Responsiveness:** At the bottom, you see @media. This tells the browser: "If the screen is smaller than a phone, hide the sidebar and show a hamburger menu instead."

### Phase 2: The Brains (JavaScript)

This is where the magic happens. The JavaScript is split into three files to keep things organized.

#### 1. data.js (The Memory)

Since this is a student project without a massive backend database (like SQL), it uses **LocalStorage**.

* **LocalStorage** is a feature in your browser that lets websites save data on your computer. It survives even if you refresh the page.
* **Seeding:** When the app loads, DataManager.init() runs. It checks: "Do we have data?" If not, it runs seedData(), which creates the fake users (Ali, Azka, Misbah), fake courses, and fake grades you see.
* **CRUD:** This file handles **C**reate, **R**ead, **U**pdate, and **D**elete operations.
  + *Example:* DataManager.grades.add() calculates the percentage and saves the grade to LocalStorage.

#### 2. utils.js (The Toolbox)

This file contains helper functions that don't care about the *business logic* but help the app run smooth.

* **Toasts:** Those little popups that say "Login Successful"? That's Utils.toast.show(). It creates a temporary HTML element, slides it in, and deletes it after 3 seconds.
* **Charts:** It uses a library called Chart.js to draw the attendance graphs.
* **PDFs:** Utils.export.generatePDF() creates a new invisible browser window, writes HTML into it, and triggers the browser's "Print to PDF" feature.

#### 3. app.js (The Manager / Controller)

This is the file that connects the User (HTML) to the Data (data.js).

**The Logic Flow of a Login:**

1. **Event Listener:** app.js waits for you to click the "Sign In" button.  
   JavaScript  
   document.getElementById('login-form').addEventListener('submit', ...)
2. **Capture:** It grabs the text from the ID and Password boxes.
3. **Authentication:** It asks data.js: "Do you have a user with this ID and Password?"
4. **Decision:**
   * **If No:** Calls Utils.toast.error("Invalid ID").
   * **If Yes:**
     + It saves the current user in the Session.
     + It runs showDashboard().
     + showDashboard() finds the <div id="login-page"> and hides it.
     + It finds <div id="dashboard-page"> and shows it.
     + It updates the HTML text document.getElementById('user-name').textContent = user.name.

### Phase 3: The Server (server/)

You might ask, "If the app runs in the browser, why is there a server folder?"

1. **Serving the App:** Browsers can't just run complex apps by double-clicking index.html perfectly (security reasons). You need a small web server to "serve" the files to the browser. That's what express and vite are doing.
2. **Sending Emails:** Browsers **cannot** send emails directly (it's a security risk).
   * When attendance drops below 85%, app.js sends a message to the *Server*.
   * The Server (server/routes.ts) receives the message and talks to an external service called **Resend API**.
   * The Resend API actually delivers the email to the student's inbox.

### Summary of "How it Works"

1. **Load:** You open the site. index.html loads. data.js checks if you have data; if not, it creates fake data in your browser's storage.
2. **Render:** app.js sees you aren't logged in, so it applies the active CSS class to the Login Page (making it visible) and removes it from the Dashboard (making it invisible).
3. **Interact:** You type credentials. app.js checks them against data.js.
4. **Switch:** If correct, app.js swaps the CSS classes. Now the Dashboard is visible.
5. **Populate:** app.js grabs your specific data (e.g., "Student") from data.js and injects it into the HTML tables and charts.
6. **Save:** If a teacher marks attendance, app.js sends that data to data.js, which saves it to LocalStorage. If you refresh, the data is still there because LocalStorage persists.

This is a professional way to build a prototype without needing a complex backend server!