**Introduction to Frontend Basics**

**Objective:** Learn HTML, CSS, and JavaScript fundamentals and create a static page with a button.

**1. What is a Frontend?**

- Explain the difference between frontend (UI) and backend (APIs/logic).

**- HTML:** Structure of a webpage (tags like `<h1>`, `<button>`).

**- CSS:** Styling (colors, layouts).

**- JavaScript:** Interactivity (click events, dynamic updates).

**2. Setup Project**

- Create a folder `frontend-day1`.

- Inside:

- `index.html` (main HTML file)

- `styles.css` (CSS file)

- `script.js` (JavaScript file)

- Use VS Code **(install [Live Server](**https://marketplace.visualstudio.com/items?itemName=ritwickdey.LiveServer**) for real-time preview).**

**HTML & Basic Structure**

**1. Code a Static Page**

**html**

<!DOCTYPE html>

<html>

<head>

<title>Day 1 Activity</title>

<link rel="stylesheet" href="styles.css">

</head>

<body>

<h1>Frontend Basics</h1>

<button id="helloButton">Click Me!</button>

<script src="script.js"></script>

</body>

</html>

**Activity:**

- Add a `<p>` tag to display messages.

- Create a list (`<ul>`, `<li>`) of your favorite programming languages.

**CSS Styling**

1. Add Styles

**css**

body {

font-family: Arial, sans-serif;

text-align: center;

margin: 2rem;

}

button {

padding: 10px 20px;

background-color: #007bff;

color: white;

border: none;

border-radius: 5px;

cursor: pointer;

}

button:hover {

background-color: #0056b3;

}

Activity:

- Change the button color to green.

- Add a border to the `<h1>` tag.

**JavaScript Interactivity**

**1. Make the Button Work**

**javascript**

**// script.js**

const helloButton = document.getElementById('helloButton');

const messageParagraph = document.createElement('p');

document.body.appendChild(messageParagraph);

helloButton.addEventListener('click', () => {

messageParagraph.textContent = 'Hello, Frontend World!';

messageParagraph.style.color = 'green';

});

**Activity:**

- Add a counter that increments each time the button is clicked.

- Add a second button to reset the counter.

**Practice Activities**

1. Create a webpage with your name, a profile image, and a hobby list.

2. Style the page with a background color and custom fonts.

3. Add a button that changes the background color when clicked.

**Key Takeaways**

- Frontend = HTML (structure) + CSS (style) + JavaScript (logic).

- Always link CSS/JS files to HTML.

- Use `document.getElementById()` and `addEventListener()` for interactivity.

**Troubleshooting Tips:**

- Check the browser console for errors (F12 > Console).

- Ensure file paths are correct (e.g., `href="styles.css"`).

This plan balances theory, hands-on coding, and creativity while keeping beginners engaged!

**CODING ACTIVITIES**

**Activity 1:** Interactive Paragraph Styling

**Goal:** Change paragraph content and style dynamically.

**HTML**

html

<!DOCTYPE html>

*<html>*

*<head>*

*<title>Paragraph Activity</title>*

*<style>*

*.highlight {*

*background-color: yellow;*

*font-size: 20px;*

*}*

*</style>*

*</head>*

*<body>*

*<p id="demoPara">Click the button to change me!</p>*

*<button onclick="changeParagraph()">Change Paragraph</button>*

*<script src="script.js"></script>*

*</body>*

*</html>*

**JavaScript (`script.js`)**

**javascript**

*function changeParagraph() {*

*const para = document.getElementById('demoPara');*

*para.textContent = "I've been updated with JavaScript!";*

*para.classList.add('highlight');*

*}*

**Task for Students:**

- Add a button to toggle the highlight on/off.

- Add a text input field to let users customize the paragraph text.

**Activity 2:** **Dynamic Table Creation**

**Goal:** Build a table using JavaScript and user inputs.

HTML

*html*

*<body>*

*<input type="text" id="nameInput" placeholder="Enter name">*

*<input type="number" id="ageInput" placeholder="Enter age">*

*<button onclick="addRow()">Add to Table</button>*

*<table id="dataTable" border="1">*

*<tr>*

*<th>Name</th>*

*<th>Age</th>*

*</tr>*

*</table>*

*<script src="script.js"></script>*

*</body>*

**CSS**

*css*

*table {*

*width: 50%;*

*margin: 20px auto;*

*border-collapse: collapse;*

*}*

*th, td {*

*padding: 10px;*

*text-align: left;*

*}*

*tr:nth-child(even) {*

*background-color: #f2f2f2;*

*}*

**JavaScript (`script.js`)**

*function addRow() {*

*const name = document.getElementById('nameInput').value;*

*const age = document.getElementById('ageInput').value;*

*const table = document.getElementById('dataTable');*

*const newRow = table.insertRow(-1); // Adds to end*

*const cell1 = newRow.insertCell(0);*

*const cell2 = newRow.insertCell(1);*

*cell1.textContent = name;*

*cell2.textContent = age;*

*}*

**Task for Students:**

- Add a button to delete the last row.

- Validate inputs (e.g., prevent empty values).

**Activity 3:** Input Validation & Styling

**Goal:** Style inputs based on validation and display messages.

**HTML**

**html**

*<body>*

*<input type="email" id="emailInput" placeholder="Enter email">*

*<button onclick="validateEmail()">Validate Email</button>*

*<p id="errorMsg" style="color: red;"></p>*

*</body>*

**CSS (Style.css)**

*input {*

*padding: 8px;*

*margin: 10px;*

*border: 2px solid #ccc;*

*}*

*input.valid {*

*border-color: green;*

*}*

*input.invalid {*

*border-color: red;*

*}*

**JavaScript(`script.js`)**

*function validateEmail() {*

*const emailInput = document.getElementById('emailInput');*

*const errorMsg = document.getElementById('errorMsg');*

*const email = emailInput.value;*

*if (email.includes('@') && email.includes('.')) {*

*emailInput.classList.add('valid');*

*emailInput.classList.remove('invalid');*

*errorMsg.textContent = '';*

*} else {*

*emailInput.classList.add('invalid');*

*errorMsg.textContent = 'Invalid email!';*

*}*

*}*

**Task for Students:**

- Add a password field with validation (e.g., minimum 6 characters).

- Show a success message when both email and password are valid.

**Bonus Activity:** **Input-to-Paragraph Mirror**

**HTML:**

*<input type="text" id="mirrorInput" placeholder="Type something">*

*<p id="mirrorText"></p>*

**JavaScript:**

document.getElementById('mirrorInput').addEventListener('input', (e) => {

document.getElementById('mirrorText').textContent = e.target.value;

});

Tips for Class

- Use `console.log()` to debug.

- Encourage experimenting with colors/fonts in CSS.

- Test edge cases (e.g., empty inputs).

Here’s a step-by-step guide with code examples to connect a Node.js/Express backend to a vanilla JavaScript frontend using HTTP requests (GET/POST):

**1.Backend Setup (Node.js/Express)**

Ensure your backend has RESTful API endpoints. Example:

**javascript**

**// server.js (Backend)**

*const express = require('express');*

*const cors = require('cors'); // For CORS*

*const app = express();*

*// Middleware*

*app.use(cors()); // Enable CORS*

*app.use(express.json()); // Parse JSON bodies*

*// Sample data*

*let students = [*

*{ id: 1, name: "Alice", age: 20 },*

*];*

*// GET all students*

*app.get('/api/students', (req, res) => {*

*res.json(students);*

*});*

*// POST new student*

*app.post('/api/students', (req, res) => {*

*const newStudent = req.body;*

*students.push(newStudent);*

*res.status(201).json(newStudent);*

*});*

*// Start server*

*app.listen(3000, () => {*

*console.log('Backend running on http://localhost:3000');*

*});*

```

2. Frontend Setup (Vanilla JavaScript)

Create a simple HTML/JS frontend to interact with the backend API:

**HTML (`index.html`):**

*<!DOCTYPE html>*

*<html>*

*<head>*

*<title>Student Manager</title>*

*<style>*

*/\* Basic styling \*/*

*body { font-family: Arial; padding: 20px; }*

*input, button { margin: 10px; padding: 8px; }*

*</style>*

*</head>*

*<body>*

*<h1>Student Manager</h1>*

*<!-- Form to add students -->*

*<input type="text" id="nameInput" placeholder="Name">*

*<input type="number" id="ageInput" placeholder="Age">*

*<button onclick="addStudent()">Add Student</button>*

*<!-- Display students -->*

*<div id="studentsList"></div>*

*<script src="script.js"></script>*

*</body>*

*</html>*

**JavaScript (`script.js`):**

Fetch and display students

*async function loadStudents() {*

*try {*

*const response = await fetch('http://localhost:3000/api/students');*

*const students = await response.json();*

*renderStudents(students);*

*} catch (error) {*

*console.error('Error loading students:', error);*

*}*

*}*

*// Add a new student*

*async function addStudent() {*

*const name = document.getElementById('nameInput').value;*

*const age = document.getElementById('ageInput').value;*

*try {*

*const response = await fetch('http://localhost:3000/api/students', {*

*method: 'POST',*

*headers: { 'Content-Type': 'application/json' },*

*body: JSON.stringify({ name, age }),*

*});*

*if (response.ok) {*

*loadStudents(); // Refresh the list*

*}*

*} catch (error) {*

*console.error('Error adding student:', error);*

*}*

*}*

*// Render students in the DOM*

*function renderStudents(students) {*

*const listDiv = document.getElementById('studentsList');*

*listDiv.innerHTML = students*

*.map(student => `<div>${student.name}, Age: ${student.age}</div>`)*

*.join('');*

*}*

*// Load students on page load*

*loadStudents();*

**3. Key Steps for Connection**

1. Start the Backend: Run `node server.js` to start the Node.js server.

2. Start the Frontend: Use a tool like VS Code’s Live Server to serve the frontend.

3. CORS: The `cors()` middleware in the backend allows the frontend to access the API.

4. HTTP Methods:

**- `fetch()`** sends requests to the backend.

**- `GET`** retrieves data.

**- `POST`** sends data (with headers and a JSON body).

**4. Testing**

- Open the frontend in a browser.

- Use Browser DevTools (**F12 > Network tab**) to monitor requests/responses.

- Test adding students and ensure the list updates dynamically.

**5.Common Issues & Fixes**

| **Issue**  | **Solution**  |

|-------------------------------- |--------------------------------------------------------------|

| CORS errors | Ensure *`app.use(cors())`* is in the backend. |

| Network errors | Check if backend is running (*`localhost:3000`*).|

| JSON parsing errors | Use `*express.json()*` middleware in backend. |

| Input validation missing | Add checks in the frontend/backend. |