**Documentation and Comments**

**1. Organized File Structure**

Ensure your project has a clear and well-organized structure with separate files for HTML, CSS, and JavaScript. An example file structure might look like this:

graphql

Copy code

Student-Registration-System/

│

├── index.html # Main HTML file

├── style.css # CSS file for styling

├── script.js # JavaScript file for functionality

└── README.md # Documentation file for the project

**2. Creativity and Presentation**

Ensure the overall presentation of the website is visually appealing. You can:

* Use a color theme that is pleasing to the eyes.
* Add some hover effects and transitions for a more interactive feel.
* Use good font choices, consistent margins, and spacing.

**3. Comments in the Code**

Ensure that you comment on the critical and complex parts of your code to explain the logic and functionality.

Example of well-commented code in script.js:

javascript

// Wait until the DOM is fully loaded

document.addEventListener('DOMContentLoaded', () => {

// Get form and table body elements

const form = document.getElementById('studentForm');

const tableBody = document.querySelector('#studentTable tbody');

// Load student data from local storage when the page is loaded

loadStudentData();

// Listen for the form submission event

form.addEventListener('submit', function (e) {

e.preventDefault(); // Prevent form from submitting normally

// Retrieve and trim the input values

const studentName = document.getElementById('studentName').value.trim();

const studentID = document.getElementById('studentID').value.trim();

const email = document.getElementById('email').value.trim();

const contactNo = document.getElementById('contactNo').value.trim();

// Validate input before adding the student

if (!validateInput(studentName, studentID, email, contactNo)) {

return; // Stop if validation fails

}

// Create a student object with the form data

const studentData = { studentName, studentID, email, contactNo };

// Add the student to the table and local storage

addStudentRecord(studentData);

// Clear the form after submission

form.reset();

});

// Function to validate form inputs

function validateInput(name, id, email, contactNo) {

const nameRegex = /^[A-Za-z\s]+$/; // Ensure only letters and spaces in name

const idRegex = /^[0-9]+$/; // Ensure only numbers in student ID

const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/; // Email validation

const contactNoRegex = /^[0-9]{10}$/; // Ensure exactly 10 digits in contact number

// Validation checks with alerts

if (!name || !nameRegex.test(name)) {

alert('Please enter a valid name (letters only).');

return false;

}

if (!id || !idRegex.test(id)) {

alert('Please enter a valid Student ID (numbers only).');

return false;

}

if (!email || !emailRegex.test(email)) {

alert('Please enter a valid email.');

return false;

}

if (!contactNo || !contactNoRegex.test(contactNo)) {

alert('Please enter a valid 10-digit contact number.');

return false;

}

return true; // Return true if all validations pass

}

// Function to add the student to local storage and display table

function addStudentRecord(studentData) {

let students = JSON.parse(localStorage.getItem('students')) || [];

students.push(studentData);

localStorage.setItem('students', JSON.stringify(students));

displayStudentRecords(); // Refresh the table display

}

// Display student records from local storage in the table

function displayStudentRecords() {

tableBody.innerHTML = ''; // Clear the table before displaying updated records

const students = JSON.parse(localStorage.getItem('students')) || [];

students.forEach((student, index) => {

const row = document.createElement('tr');

row.innerHTML = `

<td>${student.studentName}</td>

<td>${student.studentID}</td>

<td>${student.email}</td>

<td>${student.contactNo}</td>

<td>

<button onclick="editStudent(${index})">Edit</button>

<button onclick="deleteStudent(${index})">Delete</button>

</td>

`;

tableBody.appendChild(row);

});

}

// More code here for editing, deleting, etc...

});

* Add comments at the start of each major function or block of logic.
* Explain what each part of the code is doing, especially for validation, local storage handling, and event listeners.

**4. Uploading to GitHub**

To complete this task, upload your project to GitHub by following these steps:

**Steps to Upload to GitHub:**

1. Create a GitHub account if you don't have one.
2. Create a new repository:
   * Go to your GitHub dashboard.
   * Click on the "New" button to create a new repository.
   * Name it something like student-registration-system.
   * You can add a description and set it to "Public."
3. On your local machine, navigate to the folder where your project is stored.
4. Open a terminal (or command prompt) in that folder and run the following commands to initialize a git repository and push your project:

bash

git init

git add .

git commit -m "Initial commit"

git remote add origin https://github.com/yourusername/student-registration-system.git

git push -u origin master

**Link to Submit:**

Once your project is uploaded, you can provide the GitHub repository link (e.g., https://github.com/yourusername/student-registration-system).

**5. README.md Documentation**

In your GitHub repository, create a README.md file with instructions on how to use the project.

Example README.md content:

markdown

Copy code

# Student Registration System

This is a simple student registration system that allows users to add, edit, and delete student records. The data is stored using local storage, ensuring persistence even after the page is refreshed.

## Features

- Add new student records

- Edit existing records

- Delete student records

- Input validation for name, student ID, email, and contact number

- Data persistence using local storage

## How to Use

1. Clone the repository:

git clone **https://github.com/konikasingh/DomAssign/tree/main**

markdown

2. Open the `index.html` file in your browser to use the application.

## Technologies Used

- HTML

- CSS

- JavaScript

- Local Storage for data persistence