STUDENT RECORD MANAGEMENT SYSTEM

A PROJECT REPORT

Submitted by

AYUSH KAUSHIK [RA2211003011070]
MANAVI LAHOTI [RA2211003011081]
NAVYA GOEL [RA2211003011091]
SAMYAK TRIPATHI [RA2211003011095]

for the course 21CSC201J Data Structures and Algorithms

Under the Guidance of

Dr. Indhumathi Raman

Professor, Department of Computing Technologies

In partial satisfaction of the requirements for the degree of

BACHELOR OF TECHNOLOGY
in
COMPUTER SCIENCE ENGINEERING

SCHOOL OF COMPUTING

COLLEGE OF ENGINEERING AND TECHNOLOGY

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR - 603203

NOVEMBER 2023

SRM
INSTITUTE OF SCIENCE & TECHNOLOGY
Deemed to be University u/s 3 of UGC Act, 1956

SRM INSTITUTION OF SCIENCE AND TECHNOLOGY
KATTANKULATHUR-603203

BONAFIDE CERTIFICATE

Certified that the 21CSC201J Data Structures and Algorithms course project report titled "***STUDENT RECORD MANAGEMENT SYSTEM***" is the bonafide work done by **AYUSH KAUSHIK [RA2211003011070], MANAVI LAHOTI [RA2211003011081], NAVYA GOEL [RA2211003011091], SAMYAK TRIPATHI [RA2211003011095]** who carried out under my supervision.

Certified further, that to the best of my knowledge the work reported herein does not form part of any other work.

Department of Computing Technologies
Kattankulathur
603 203
SRM IST

[illegible]

Dr. Indhumathi Raman,
Professor, Faculty In-Charge,
Department of Computing Technologies,
SRMIST.

M. Pushpalatha

Dr. M. Pushpalatha,
Professor and Head,
Department of Computing Technologies,
SRMIST.

PROGRAM DEFINITION

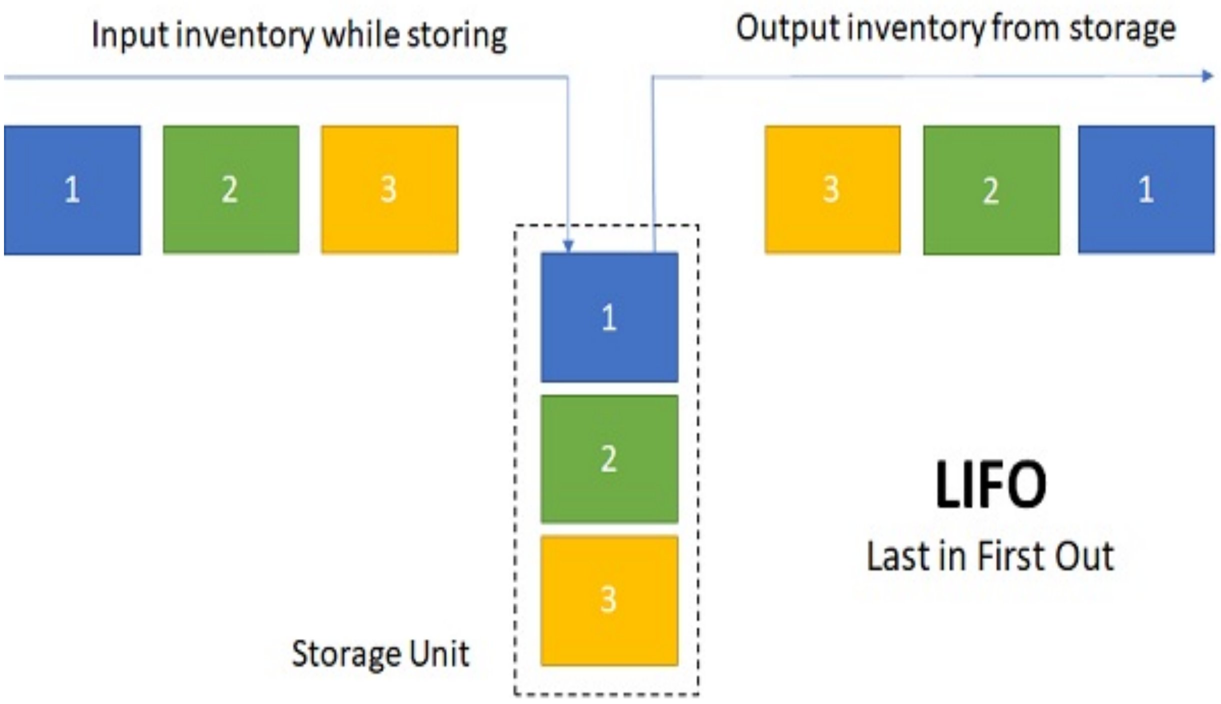
Educational institutions grapple with manual and error-prone student record management, leading to inefficiencies and accessibility challenges. Limited transparency in communication and fragmented processes hinder effective collaboration. Class scheduling issues and financial management complexities further compound administrative burdens. Existing systems lack analytical tools, impeding informed decision-making. Security concerns and integration challenges pose risks to sensitive student data. To address these issues, a Student Record Management System (SRMS) is crucial. It automates data entry, streamlines processes, and ensures accuracy in academic records. Enhanced accessibility facilitates timely decision-making and transparent communication among stakeholders. Improved class scheduling and financial management optimize resource allocation. Robust analytics provide valuable insights for institutional growth. Strengthened security measures protect sensitive information, and seamless integration with other systems promotes a cohesive educational ecosystem. The SRMS aims to transform administrative efficiency, transparency, and collaboration in managing student information within educational institutions.

PROBLEM EXPLANATION

In the realm of education, the reliance on manual student record management systems has given rise to a host of challenges. The arduous nature of data entry and retrieval processes introduces inefficiencies and errors, consuming valuable time and compromising the accuracy of student information. Limited accessibility to crucial data impedes timely decision-making and collaboration among students, teachers, and administrators. Communication breakdowns and fragmented systems exacerbate these issues, leading to misunderstandings and a lack of transparency. The complexities of class scheduling and financial management, prone to errors and conflicts, further disrupt the seamless organization of academic activities. Additionally, the absence of robust analytical tools deprives institutions of meaningful insights from student data, hindering strategic decision-making and institutional growth.

Compounding these challenges are security vulnerabilities that expose sensitive student information to potential breaches, compromising privacy and confidentiality. Integration issues with other institutional systems create information silos, impeding the seamless flow of data across different facets of the educational environment. In summary, the current manual student record management system falls short in terms of efficiency, accuracy, accessibility, and security, necessitating the immediate implementation of a comprehensive Student Record Management System to address these pressing issues and foster a more streamlined and secure educational administration.

In our project, the array `studentRecords` is used as a simple stack data structure to manage student records. A stack is a last-in, first-out (LIFO) data structure, meaning that the last element added is the first one to be removed. Here's a more detailed breakdown of how the stack concept is implemented in your code:



LIFO, which stands for Last In, First Out, is a fundamental principle of stack data structures. In a LIFO structure, the last item that is added to the stack is the first one to be removed. Imagine a stack of plates: you add a new plate to the top, and when you want to take a plate, you take it from the top. Similarly, in programming and data structures, the last element added to a stack is the first one to be accessed or removed. This Last In, First Out behavior simplifies the management of data, making it particularly useful for scenarios where the order of operations or elements is crucial, such as in undo functionality or managing function calls.

1. Push Operation (Create Record):

   The `createRecord` function utilizes the `push` method to add a new student record to the end of the `studentRecords` array. This operation simulates pushing a new element onto the stack.

2. Pop Operation (Create Record):

   The `deleteRecord` function uses the `pop` method to remove the last element from the `studentRecords` array, effectively simulating a pop operation from the stack. This operation deletes the most recently added student record.

3. Access and Search:

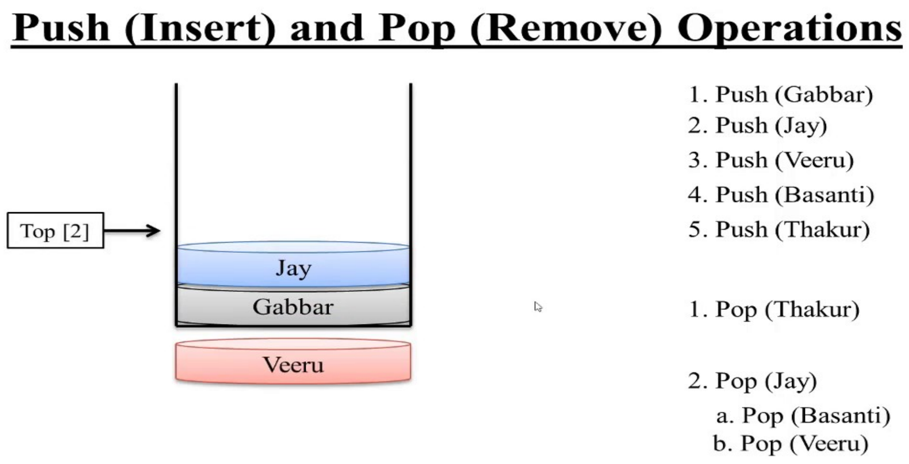
   While not explicitly demonstrated in your provided code, a common stack operation is peeking, which involves accessing the top element without removing it. In a stack-like structure, you could implement a function to peek at the top student record without removing it from the array.

4. Viewing the stack (View all records):

   The `viewAllRecords` function iterates through the `studentRecords` array, displaying the details of each student record. This operation provides a way to view the entire stack of student records, showcasing the LIFO order.

In summary, our code leverages an array as a stack to manage student records, with operations like creating, deleting, and viewing records following the principles of a last-in, first-out data structure. This stack-like structure provides a straightforward way to manage the state of student records and perform operations in a controlled and predictable manner.

Taking an example of our implementation, the `push` and `pop` functions in our javascript codes are fundamental operations for managing the `studentRecords` array, which acts as a stack. The `createRecord` function uses the `push` method to add a new student record to the end of the `studentRecords` array. This operation corresponds to pushing a new element onto the stack. The `push` method appends the newly created student record to the top of the "stack," maintaining the Last In, First Out (LIFO) order.



The `deleteRecord` function utilizes the `pop` method to remove the last element from the `studentRecords` array. This simulates a pop operation from the stack, where the most recently added student record is removed. The `pop` method returns the removed element, but in this code, the returned value is not used, as the primary purpose is to mimic the removal of the most recently added record.

These operations align with the principles of a stack data structure, allowing for the dynamic addition and removal of student records. The `push` function adds records to the top of the stack, and the `pop` function removes the last-added record, following the Last In, First Out behavior inherent in stack structures. This simple and intuitive approach to managing data is useful for scenarios like maintaining a history of actions or changes, as demonstrated in the provided code.

DATA STRUCTURE USED

The primary data structure employed is stack used as an array, specifically the `studentRecords` array, which serves as a simplified representation of a stack. A stack, following the Last In, First Out (LIFO) principle, is a linear data structure where elements are added and removed from the same end. In the context of this code, the array facilitates the management of student records. The `push` method is utilized to add new student records to the end of the array, effectively mimicking a push operation onto the stack. Conversely, the `pop` method removes the last element from the array, simulating a pop operation from the stack and enabling the deletion of the most recently added student record. This array-based implementation of a stack offers a straightforward approach to managing student data and aligns with the inherent LIFO behavior of stack structures.

The array data structure in this code provides simplicity and flexibility. Arrays in JavaScript can dynamically resize, accommodating varying numbers of student records. Additionally, the array allows for the representation of more complex structures, with each student record being an object. While not a traditional stack with a fixed size, this array-based approach effectively captures the essence of a stack for managing student records in a concise and adaptable manner.

The chosen array data structure offers both simplicity and flexibility in managing student records. Arrays in JavaScript dynamically resize, accommodating varying numbers of records, and can represent more complex structures, with each student record being an object. Despite not strictly adhering to a traditional stack with a fixed size, this array-based approach aptly embodies the essence of a stack, offering an intuitive and adaptable solution for handling student data. The array's inherent features align with the LIFO behavior, simplifying the implementation of operations such as creating and deleting records while providing a versatile platform for future enhancements to the student management system.

IMPLEMENTATION

HTML

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

<div id="container">
  <h1>Welcome to Student Record Management System</h1>
  <div id="options">
    <button onclick="createRecord()">Create a new Record</button>
    <button onclick="deleteRecord()">Delete a student record</button>
    <button onclick="searchRecord()">Search a Student Record</button>
    <button onclick="viewAllRecords()">View all students record</button>
    <button onclick="exit()">Exit</button>
  </div>

  <div id="output"></div>
</div>

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

CSS:

```
body {
    font-family: 'Poppins', monospace;
    background-color: black;
    color: white;
    margin: 0;
    padding: 0;
}

#container {
    max-width: 800px;
    height: 500px;
    margin: 90px auto;
    background-color: black;
    padding: 20px;
    align: bottom;
    border-radius: 8px;
    box-shadow: 0 0 20px rgba(0, 0, 0, 0.1);
    transition: box-shadow 0.3s ease;
}

#container:hover {
    box-shadow: 50 20 30px rgba(255, 0, 0, 0.2);
}
body {
    background-image: url('chevrolet-camaro-8k-2560x1440-13233.jpg'); /* Replace 'your-
image-url.jpg' with the actual URL or path to your image */
    background-size: cover; /* This property ensures that the background image covers the
entire container */
    background-position: top; /* This property centers the background image */
    background-repeat: no-repeat; /* This property prevents the background image from
repeating */
```

```
  }

h1 {
    text-align: center;
    color: turquoise;
    margin-bottom: 50px;
    font-size: 30px;
}

#options {
    text-align: center;
    margin-bottom: 20px;
}

button {
    padding: 18px 20px;
    margin: 8px;
    cursor: pointer;
    font-size: 18px;
    font-weight: bold;
    background-color: #e9a30d;
    color: #000000;
    border: none;
    border-radius: 10px;
    transition: background-color 0.8s ease;
}

button:hover {
    background-color: #45a049;
}
```

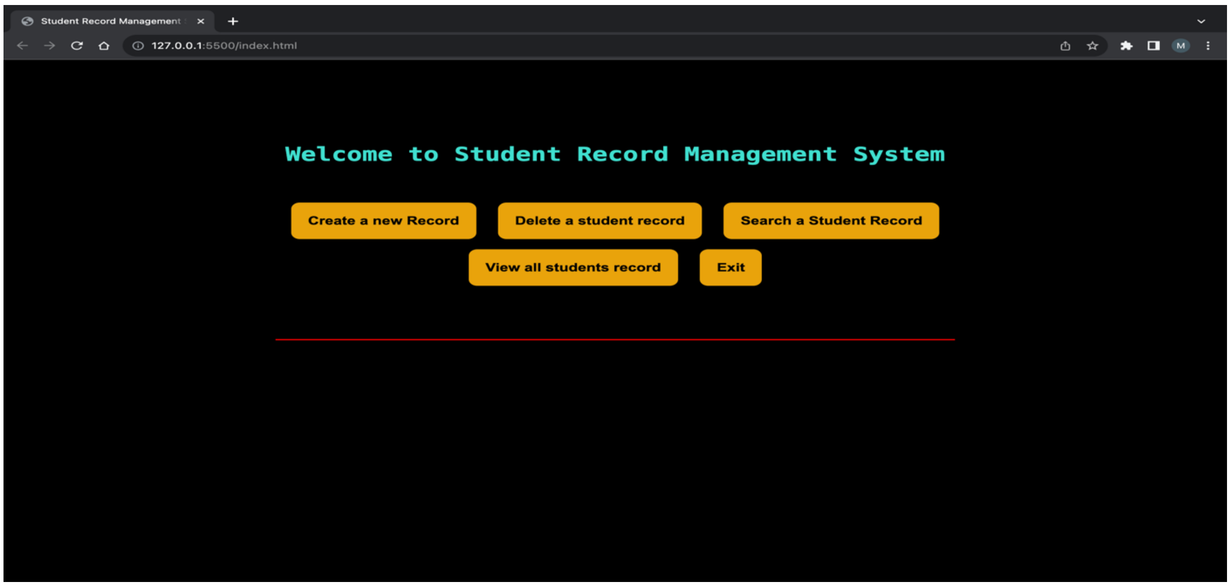
```
#output {
    border-top: 2px solid #f10000;
    margin-top: 75px;
    padding-top: 75px;
    text-align: left;
    word-spacing: 10px;
    font-size: 25px;
    color:#45a049;
}

/* Improved popup styles with animation */
.popup-container {
    display: none;
    position: center;
    top: 0;
    left: 0;
    width: 100%;
    height: 100%;
    background: rgba(0, 0, 0, 0.5);
    justify-content: center;
    align-items: center;
    z-index: 1000;
    animation: fadeIn 0.3s ease;
}

.popup {
    background: #fff;
    padding: 20px;
    border-radius: 8px;
    box-shadow: 0 0 20px rgba(0, 0, 0, 0.2);
    animation: slideIn 0.3s ease;
}

@keyframes fadeIn {
    from {
      opacity: 0;
    }
    to {
      opacity: 1;
    }
}

@keyframes slideIn {
    from {
      transform: translateY(-20px);
      opacity: 0;
    }
    to {
      transform: translateY(0);
      opacity: 1;
    }
}
```

JavaScript:

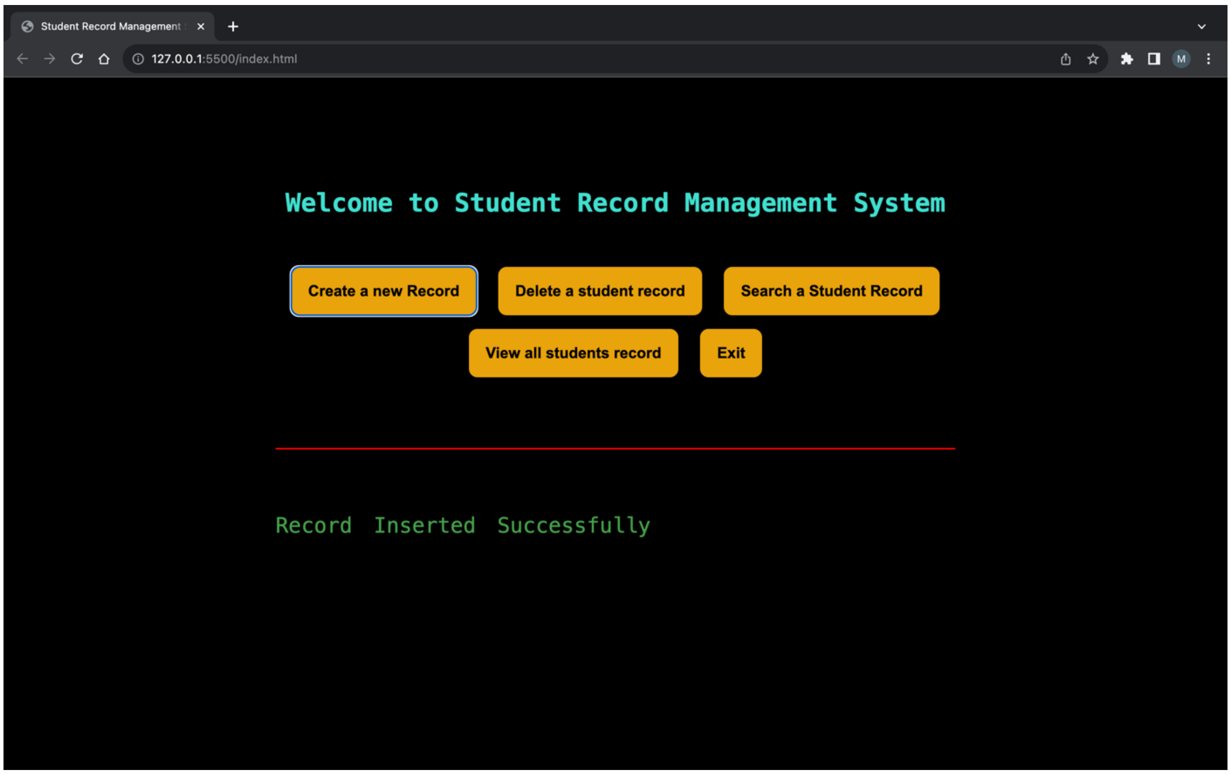
```
// Student records stored in JavaScript array
let studentRecords = [];

function createRecord() {
    const name = prompt("Enter Name of Student:");
    const roll = prompt("Enter Roll Number of Student:");
    const dept = prompt("Enter Department of Student:");
    const marks = prompt("Enter Total Marks of Student:");

    // Create a new student record object
    const student = {
        roll: parseInt(roll),
        name: name,
        dept: dept,
        marks: parseInt(marks)
    };

    // Add the new record to the array
    studentRecords.push(student);

    displayOutput("Record Inserted Successfully");
}

function deleteRecord() {
    if (studentRecords.length === 0) {
        displayOutput("No records to delete");
        return;
    }
    // Remove the last record from the array (simulating pop operation)
    studentRecords.pop();

    displayOutput("Record Deleted Successfully");
}

function searchRecord() {
    if (studentRecords.length === 0) {
        displayOutput("No records to search");
        return;
    }

    const roll = prompt("Enter Roll Number of Student:");

    // Search for the record with the given roll number
    const foundRecord = studentRecords.find(record => record.roll === parseInt(roll));

    if (foundRecord) {
        displayOutput(`Roll Number: ${foundRecord.roll}\nName:
${foundRecord.name}\nDepartment: ${foundRecord.dept}\nMarks: ${foundRecord.marks}`);
    } else {
        displayOutput("No such Record Available");
    }
}

function viewAllRecords() {
    if (studentRecords.length === 0) {
        displayOutput("No Record Available");
        return;
    }
    let output = "Index\t\tName\t\tCourse\t\tMarks\n\n";
    studentRecords.forEach((record, index) => {
        output += `${index + 1}\t${record.name}\t${record.dept}\t${record.marks}\n`;
    });

    displayOutput(output);
}

function exit() {
    alert("Exiting the application");
}

function displayOutput(message) {
    const outputDiv = document.getElementById("output");
    outputDiv.innerText = message;
}
```

OUTPUT

Homepage

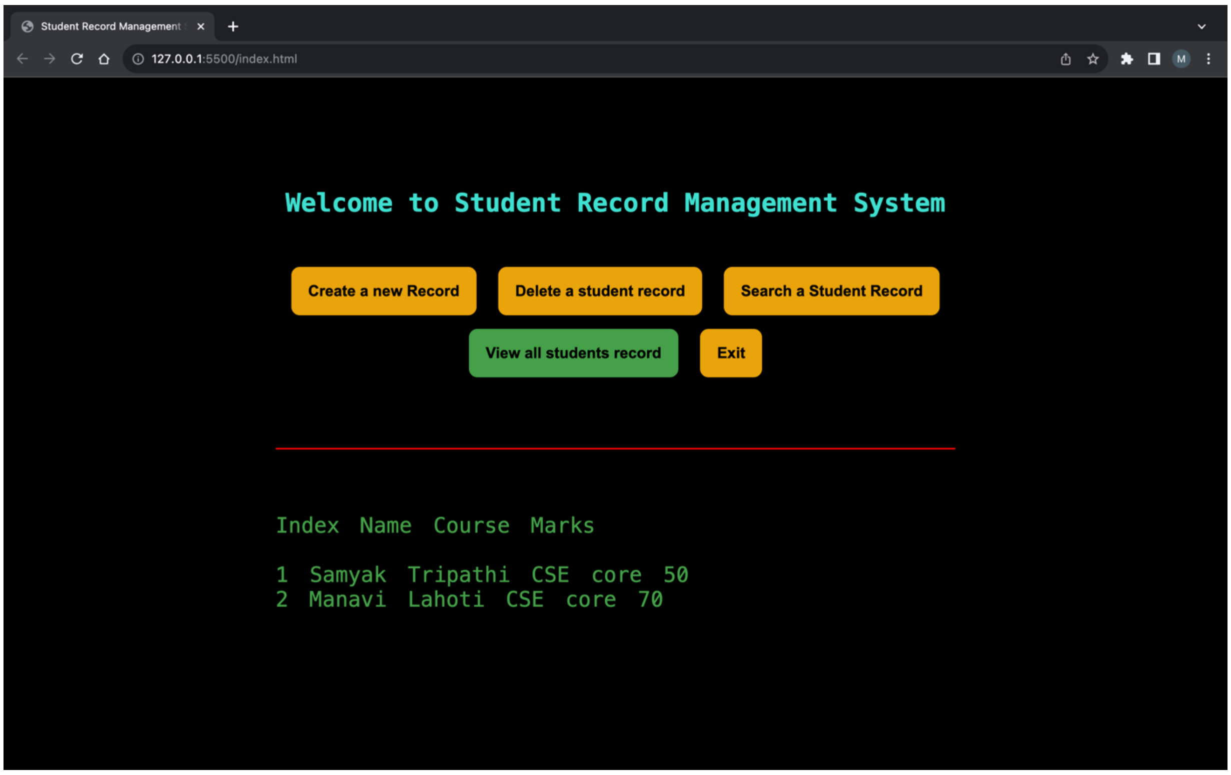

Student Record Management
127.0.0.1:5500/index.html
Welcome to Student Record Management System
Create a new Record
Delete a student record
Search a Student Record
View all students record
Exit

Create a record:


Student Record Management
127.0.0.1:5500/index.html
Welcome to Student Record Management System
Create a new Record
Delete a student record
Search a Student Record
View all students record
Exit

```
Record  Inserted  Successfully
```

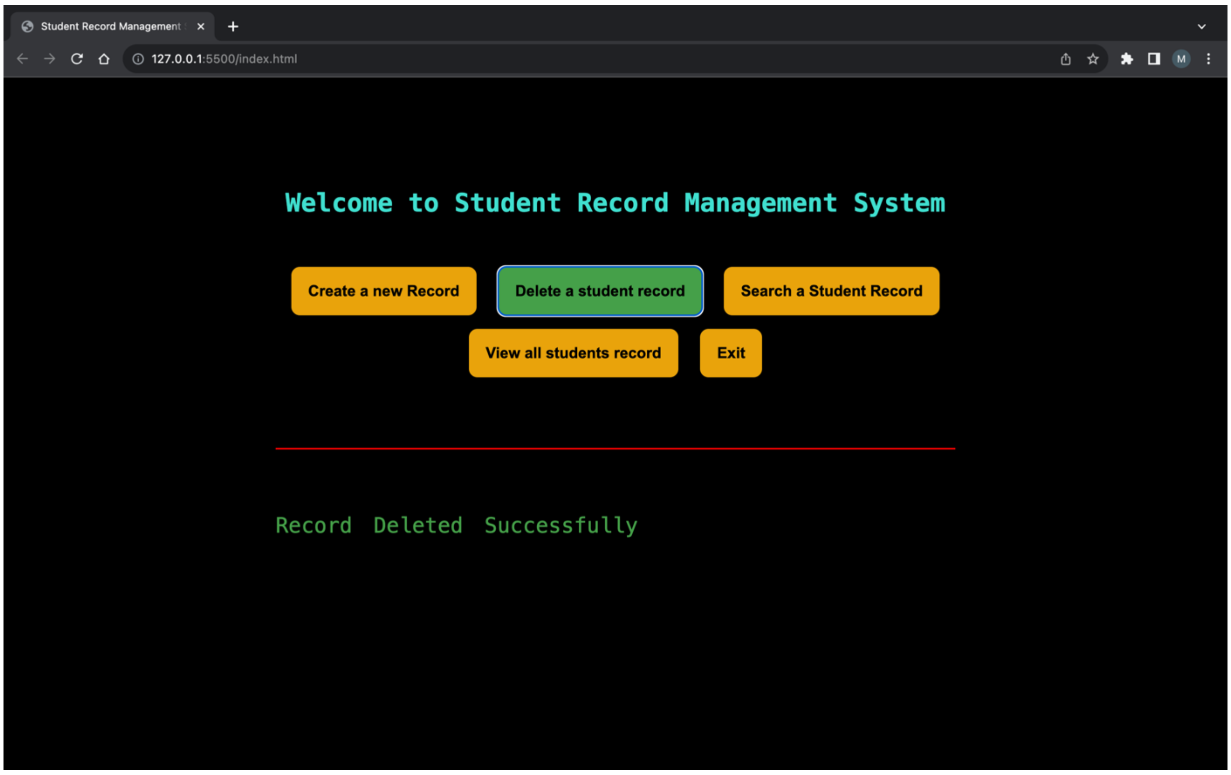
View Records:



Welcome to Student Record Management System

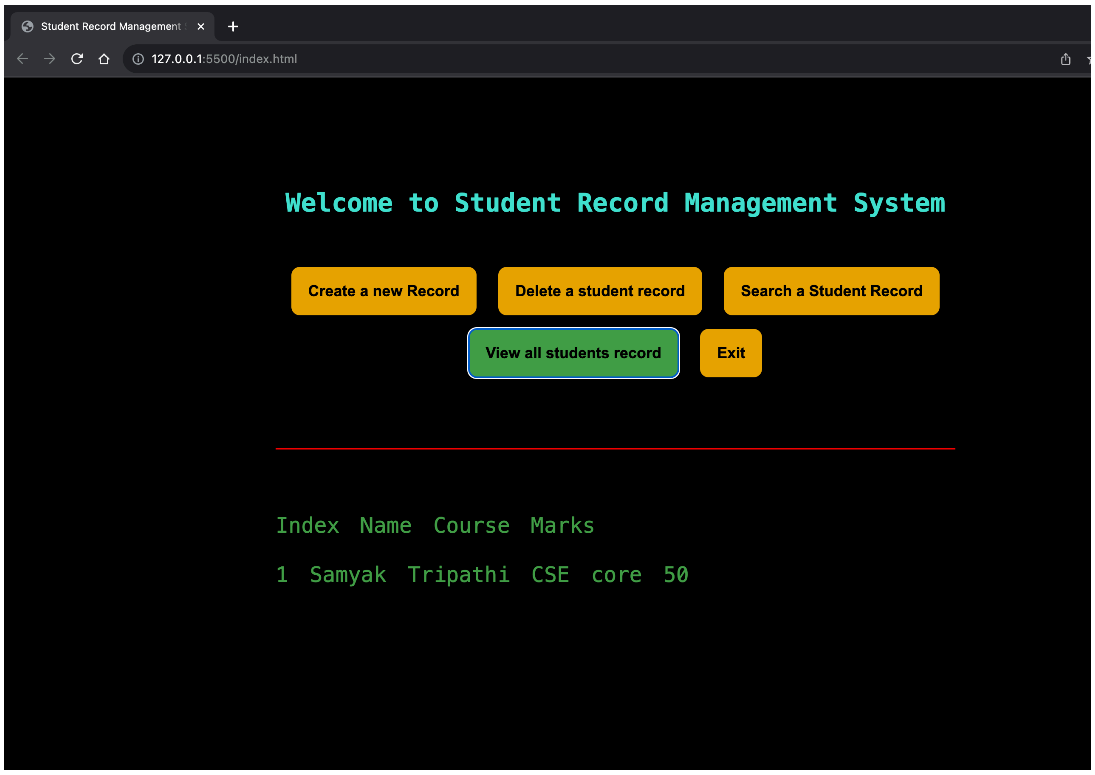
```
Index  Name  Course  Marks

1  Samyak  Tripathi  CSE  core  50
2  Manavi  Lahoti  CSE  core  70
```

Delete Record:



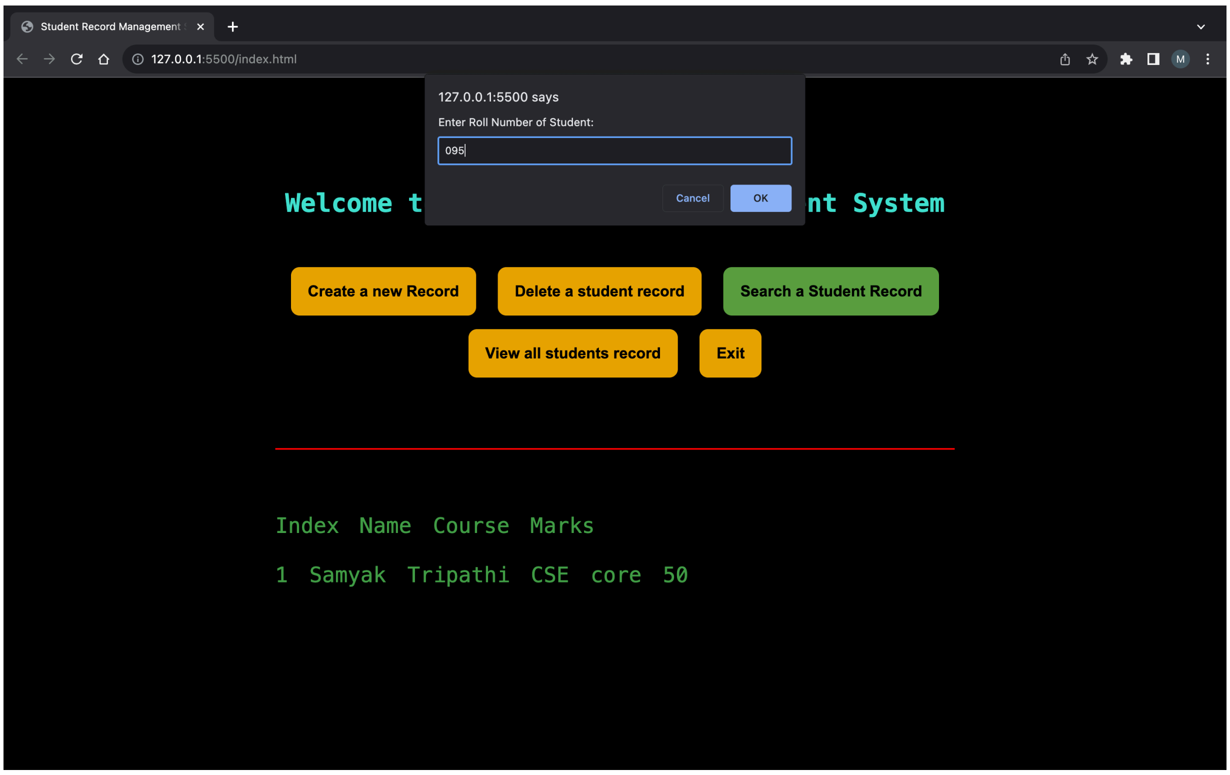
Welcome to Student Record Management System

```
Record  Deleted  Successfully
```

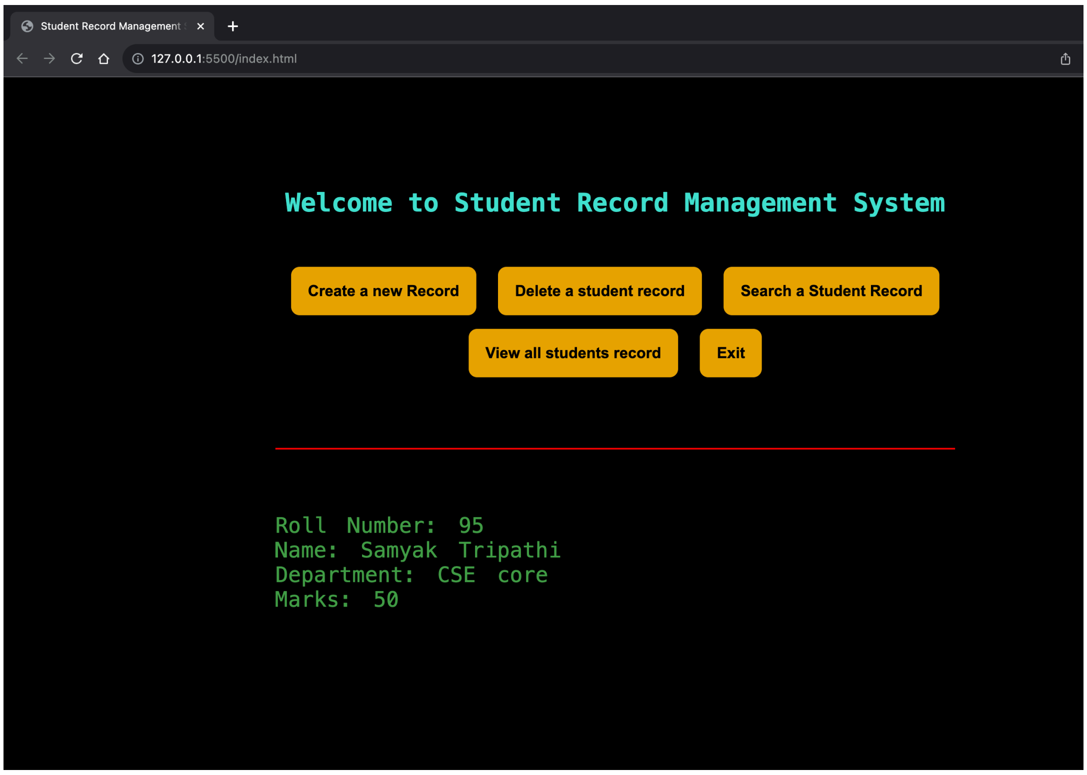

Student Record Management
127.0.0.1:5500/index.html
Welcome to Student Record Management System
Create a new Record
Delete a student record
Search a Student Record
View all students record
Exit

```
Index  Name  Course  Marks

1  Samyak  Tripathi  CSE  core  50
```

Search a Record:



```
Index  Name  Course  Marks

1  Samyak  Tripathi  CSE  core  50
```


Student Record Management
127.0.0.1:5500/index.html
Welcome to Student Record Management System
Create a new Record
Delete a student record
Search a Student Record
View all students record
Exit

```
Roll  Number:  95
Name:  Samyak  Tripathi
Department:  CSE  core
Marks:  50
```

CONCLUSION

In conclusion, the JavaScript code presented for the student management system effectively leverages the array data structure to emulate a stack, implementing basic stack operations such as push and pop. The stack-like structure allows for the orderly management of student records, with the Last In, First Out (LIFO) behavior governing the addition and removal of records. This approach provides a simple and intuitive way to handle student data, supporting operations like creating, deleting, and viewing records.

While the current implementation focuses on essential stack functionalities, further enhancements could be considered. For instance, incorporating additional features such as undo functionality by maintaining a history of operations, implementing error handling, or connecting the system to a persistent data storage solution could enhance the robustness and functionality of the student management system.

In summary, the project lays a foundation for managing student records in a stack-like structure, demonstrating the fundamental principles of a stack data structure. Future development and refinement could extend the capabilities of the system, ensuring its adaptability and usefulness in a broader educational context.