PROJECT

Report on

ONLINE STUDENT REGISTRATION SYSTEM

by

Ambrish Shukla (2000290140017)
Abhinandan Saini (2000290140004)
Aaditya Singh (2000290140001)

Session:2021-2022 (4th Semester)

Under the supervision of

Ms. Divya Singhal (Assistant Professor)

KIET Group of Institutions, Delhi-NCR, Ghaziabad

DEPARTMENT OF COMPUTER APPLICATIONS
KIET GROUP OF INSTITUTIONS, DELHI NCR,
GHAZIABAD-201206
(MAY- 2022)

CERTIFICATE

Certified that Ambrish Shukla (2000290140017) , Abhinandan Saini (2000290140004), Aaditya Singh (2000290140001) has carried out the project work presented in this report entitled "ONLINE STUDENT REGISTRATION SYSTEM" for the award of Master of Computer Application from Dr. A.P.J. Abdul Kalam Technical University, Lucknow under my supervision.The report embodies result of original work, and studies are carried out by the students themselves and the contents of the report do not form the basis for the award of any otherdegree to the candidate or to anybody else from this or any other University.

Ms Divya Singhal

Assistant Professor

Dept. of Computer Applications

KIET Group of Institutions, Ghaziabad

External Examiner

Dr. Ajay Kumar Srivastava

Professor & Head

Department of Computer Applications

KIET Group of Institutions, Ghaziabad

Date: 28 may 2022

ABSTRACT

This is a website of Online Student Registration system which can be used by student to register themselves. This website of Student Registration removes the task of student and management for registration in admission in college.

This registration system will help the administrative staff that is executive of the college/school to keep the daily and the history record details of the student within the proper database. The system aims at the registration and management of the Colleges/Schools that are available in the College. It mainly takes care of the Students in the database.

The system provides the information regarding the students that are available and their status specific to availability. The students can visit the site with the necessary information that is expected by the management from students.

Each registered student can see their details on site. The total front end was developed by using HTML & CSS standards and for dynamic effects we have applied the JAVASCRIPT is one of the secure languages that is planned for the database connectivity. The user level accessibility has been restricted into two zones the administrative and the normal user zone.

ACKNOWLEDGEMENT

We take this occasion to thank God, almighty for blessing us with his grace and taking our endeavor to a successful culmination. We extend my sincere and heartfelt thanks to our esteemed guide, Ms Divya Singhal, for providing us with the right guidance and advice at the crucial junctures and for showing us the right way. We extend our sincere thanks to our respected Head of the department Dr. Ajay Kumar Shrivastava, for allowing us to use the facilities available. We would like to thank the other faculty members also, at this occasion. Last but not the least, we would like to thank my friends and family for the support and encouragement they have given us during our work.

Ambrish Shukla (2000290140017)

Abhinandan Saini (2000290140004)

Aaditya Singh (2000290140001)

TABLE OF CONTENTS

DECLARATION

We hereby declare that the work presented in this report entitled "ONLINE STUDENT REGISTRATION SYSTEM", was carried out by us.

We have not submitted the matter embodied in this report for the award of any other degree or diploma of any other University or Institute.

We have given due credit to the original authors/sources for all the words, ideas, diagrams, graphics, computer programs, experiments, results, that are not my original contribution.

We have used quotation marks to identify verbatim sentences and given credit to the original authors/sources.

We affirm that no portion of my work is plagiarized, and the experiments and results reported in the report are not manipulated.

In the event of a complaint of plagiarism and the manipulation of the experiments and results,

We shall be fully responsible and answerable.

Name: Ambrish Shukla

Abhinandan Saini

Aaditya Singh

Roll. No.: 2000290140017

2000290140004

2000290140001

Branch: Master of Computer Application

<u>Introduction</u>

System may be defined as a layered structure that depicts how programs involved would interrelate and communicate. In computers, System may also include actual programs, programming interfaces and tools for managing the larger system. The term system may be used differently in different contexts, but the concept remains the same. Online student registration system combines multiple systems to construct a combined framework. This framework consists of multiple modules, which further contain different systems along with the implementation of their defined constraints.

Basically, systems are implemented for facilitating complex manual processes and that is exactly what we are trying to achieve. System is implemented as per user requirement such as a manufacturing concern may install a plant for easing out manual processes. We have sought help from computer programming for automation of manual registration system.

With the introduction of computers, every aspect of our lives has been revolutionized. When used judiciously, computers can help us save time, secure our personal information, access the required information whenever and wherever required. Keeping all these positive points in mind, we have developed an Online Student

Registration System for easily managing the semester registration process for the student in an institution. Ours is an advisory based system. In state agricultural universities the allocation is advisory based and more complicated. These are assigned according to the skill set and industry requirements. Hence, in current scenario, automated system is required for registration of students.

Implementation of the proposed system will reduce the workload of all those involved as the data can be now managed with proper authentication and authorizations instead of being hard copied and accessible to everyone. This system will largely save the precious time of Deans, Advisors and Accounts Officers, instead of explicitly signing every document; they just must acknowledge entries online with the click of a mouse.

All the technologies i.e., PHP, Apache and MySQL used for current system design are open source and hence freely available for download. PHP provides a strong platform for creating the visual front-end of the web application and PHP combined with HTML provides a very flexible development environment.

For simpler implementation of certain features. For constant testing, analysis and execution needs, Firefox and Google Chrome web-browsers were used. With a combination of all these technologies we were able to create a web application environment that is efficient and consistent enough.

The primary goal of our research and development was to automate student registration procedure. It has been achieved successfully and the system is tested to be working efficiently. Online application of the whole system helps easy access to the system anywhere. Physical presence of the student is not required. The time taken for process completion is now largely reduced. After registration the database is automatically updated at the end of process completion removing the hassle for department officials who had to enter the data manually. As the database is managed through MySQL, data duplication is eliminated and thereby reducing chances of error. Also, data can now be easily retrieved, edited, and printed whenever required. Authentication based access proves to be more secure than manual system. The data is maintained on a central server and is distributed among different departments as per requirement and copies of this database are maintained on backup servers. Also, database access is authorized and cannot be viewed or edited by unauthorized personnel. So, this automated and computerized system is safe,fast and user friendly

Literature Review

According to Engr. P. D Joseph (2006) said that there was a time in the primitive and barbarian days before computer, the amount of information shepherded by a group of people could be collected in the wisdom and the stories of its older members. In this world story tellers, magicians and grandparents were considered a great and honored storehouse for all that was known.

It gets to a stage when the data are too much to be managed in the minds of the elders. And so, to store all the new information, humanity invented the technology of writing and then great scholars like Aristotle warned that the invention of the alphabet would lead to the subtle but total demise of the creativity and sensibility of humanity, data began to be stored in voluminous data repositories called books.

As we know, eventually books capsulated with great speed and soon whole communities of books migrated to the first real “database” libraries. Unlike previous versions of data warehouses (i.e., People and books) that might be considered the australopitheaic of the database lineage, libraries crossed over into the modern-day species, though they were incredibly primitive of course over into libraries introduced.

Objective

The current research aims at reducing the workload all the entities involved in the registration procedure for the students. The current manual system faces different challenges as to maintaining data of each student manually.

Hard copy registers are maintained currently to verify student details. From students' point of view, they must fill the forms manually and then get them verified from concerned officials, which is a very time-consuming process.

The objectives of this proposed web application system are:

- To computerise student and faculty database.
- To maintain data consistency.
- Automate the registration process without any physical human interaction
- Making the registration process accessible anywhere to the student.
- Allowing faculty to acknowledge registration requests from anywhere.

With the requirement of registration process for every semester, it becomes even more important to simplify a process which is highly repetitive.

The achievement of the above objectives can help the institution in managing the resources efficiently. The automated process will lead to time saving and eradication of common errors.

Research Methodology

An individual corporation bodies or even a nation we are confronted with a lot of problems everybody such as relaxing to education technology, physiology, and psychological aspect of life.

To solve these problems, we must make a strong decision as to methods and steps of solving the various problems. To be able to make a headway we need to conduct research.

Therefore, research is considered as the process of arriving at a dependable solution to a given problem through the systematic collection, analysis, and interpretation of data.

Method of Data Collection

There are numbers of approach to data collection depending on the nature of the research being conducted. In this project, the methods adopted include the following: Interview, World Wide Web, references to published and unpublished collection. The data collected for this research can be broadly classified into two types, namely: the primary and secondary data.

PRIMARY DATA

Primary data can be defined as data collected directly from respondent relevant to the subject under investigation. The primary data used in this case is interview method according to Enr. D. O Dimoji (2022) says that primary source data collection is source from first-hand information can be obtained. The tools for gathering the primary source of data collection include interview, observation, and questionnaire etc.

In outcome we gather some data from end user for further process and verify and then validate it may be defined as the fundamental data which is needed for registration.

In this registration process there is need to have a personal email id which is valid format and

- He/she needs to a password which helpful for login

- Through password she can login in the site and able to verify his or himself so they have to create a password for the login and after creating the password they will enter in the next module so this is the fundamental data which the user need in this case the end user is that the student which is trying to register itself for the course it may be the courses like MCA and other courses like MBA or B Tech after then login the next phase is started here

- Now in the next phase we need to add some students extra details like students first name and then in second column students middle name it will be the optional for the student to seal it or not and after the middle name he needs to be registered with his last name or which we can call it's a surname after entering the surname and last name or middle name in it's to enter the mobile number for the verification and contact

- Now in the next box needs to enter this father or mother name his father's first name and surname also his mother's first name and surname in both names middle name is optional and it is up to the student he/she wants to enter or not after that he or she needs to be entered his city as address is needed for the registration in the college admission process It's to be entered in both addresses current and permanent address.

- after his student in need to enter his previous educational details like 10th, 12th, and graduation, if they are applying for that post-graduation course Like MCA and MBA but if they apply for the BTech there is no need for graduation details.

Outcome

Implementing the Online Student Registration System, the registration procedure has been simplified. Previously student had to go door to door to get the documents acknowledged from the concerned officials whereas the currently developed system offers an efficient way to perform these operations.

The students can access the registration portal online either from a computer or a smart phone and fill the necessary information and submit it for further approval. This web application provides us with ease of access, user friendliness and transparency. On the other hand, from organizations viewpoint, it helps in maintaining transparency, data consistency, data accessibility and easy maintenance.

A. **It will automate.**
B. **All process will be faster.**
C. **Easy to the students to register themselves.**
D. **Can register remotely.**
E. **Without any physical contact.**
F. **Time saving.**

<u>SRS(SOFTWARE REQUIREMENT SPECIFICATION)</u>

3.1. DATA FLOW DIAGRAM(DFD)

A DFD also known as 'bubble chart, has the purpose of clarifying system requirements and identifying major transformations. It shows the flow of data. through a system. It is a graphical tool because it presents a picture. Four simple notations are used to complete a DFD. These notations are given below:

DATA FLOW

The data flow is used to describe the movement of Information from one part of the system to another part. Flows represent data in motion. It is a pipe line through which information flows. Data flow is represented by an arrow.

PROCESS

A circle or bubble represents a process that transforms incoming data to outgoing data. Process shows a part of the system that transform inputs to outputs.

EXTERNAL ENTITY

A square defines a source or destination of system data. External entities represent any entity that supplies or receive information from the system but is not a part of the system.

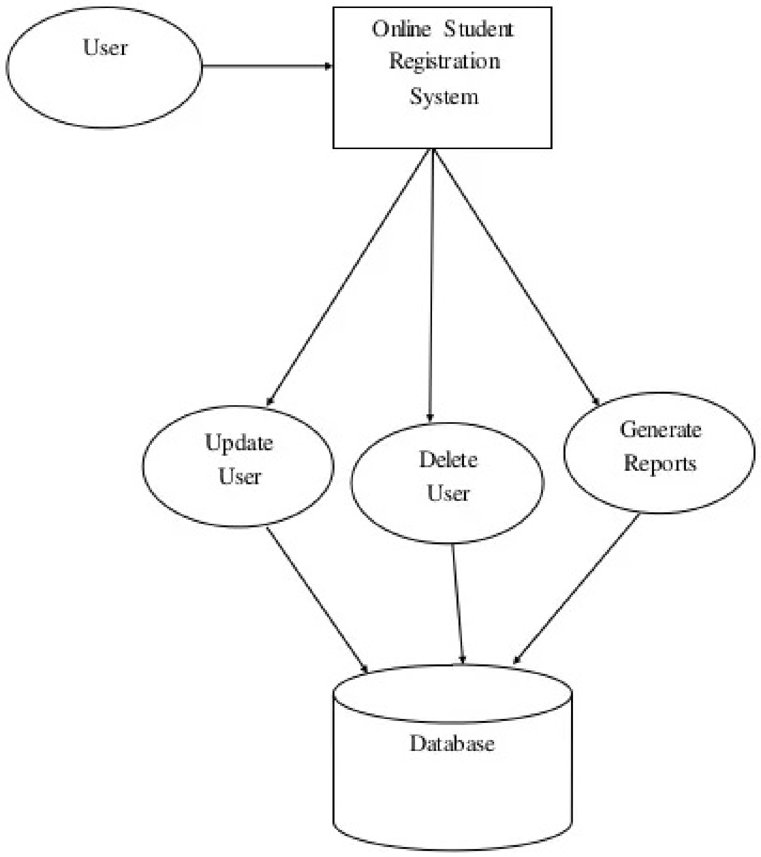
DATA STORE

The data store represents a logical file. A logical file can represent either a data store symbol which can represent either a data structure or a physical file on disk. The data store is used to collect data at rest or a temporary repository of data. It is represented by open rectangle.

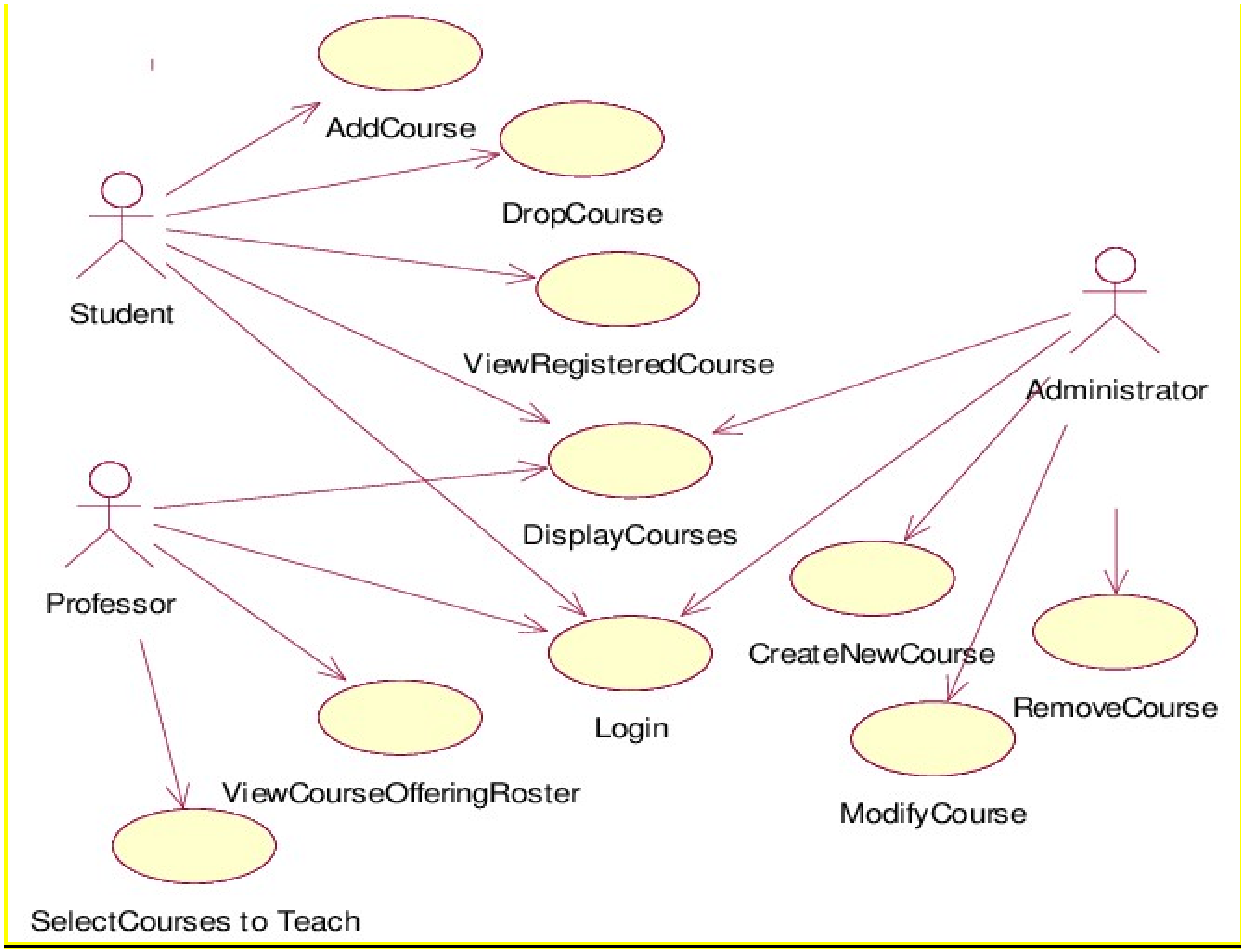
OUTPUT

The output symbol is used when a hard copy is produced and the user of the copies cannot be clearly specified or there are several users of the

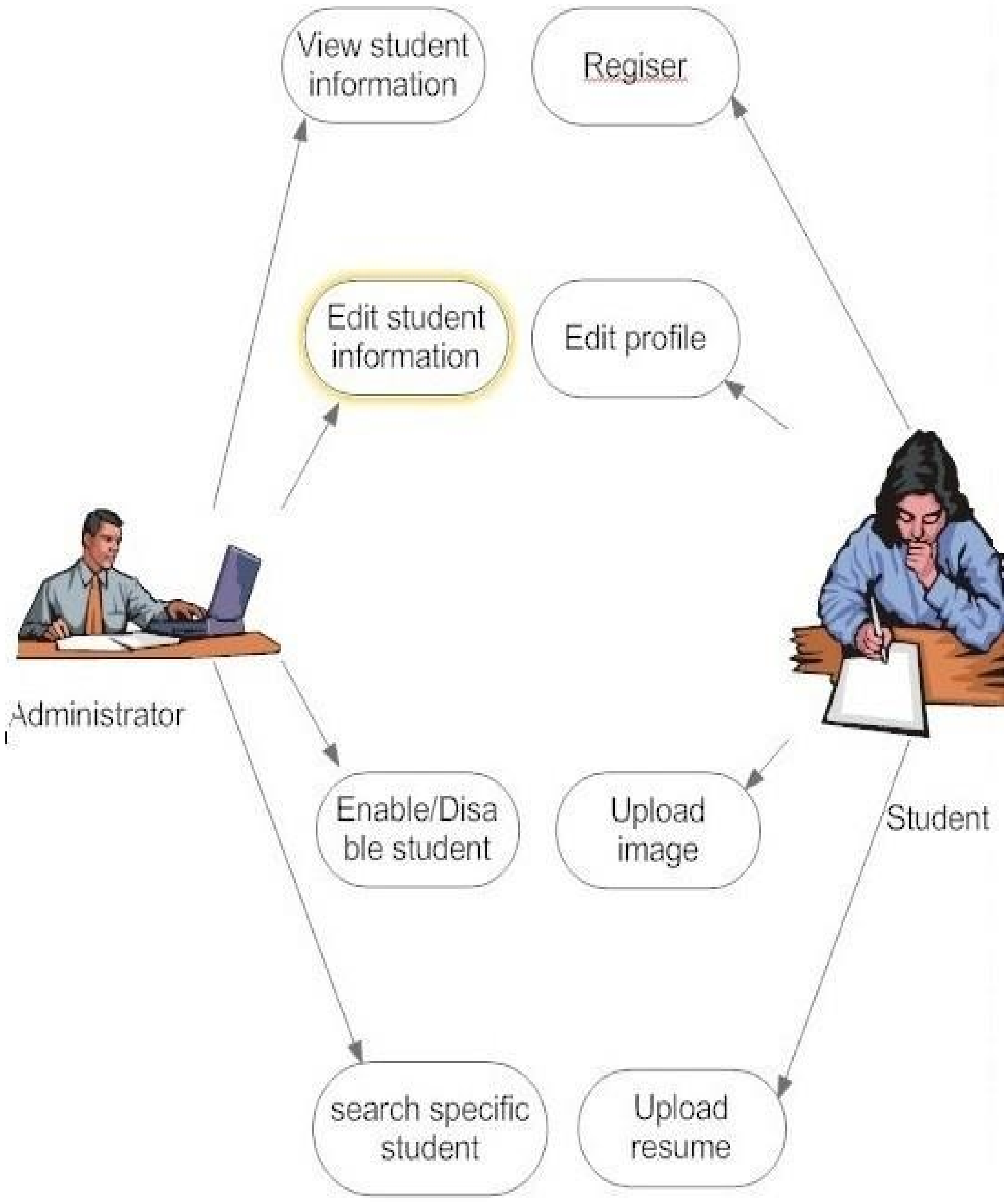
LEVEL- 1 DFD For Administrator/User



USE Case Diagram


AddCourse
DropCourse
Student
ViewRegisteredCourse
Administrator
DisplayCourses
Professor
CreateNewCourse
RemoveCourse
Login
ViewCourseOfferingRoster
ModifyCourse
SelectCourses to Teach

USE CASE MODEL



Time Duration

PROJECT TASK	MARCH	APRIL	MAY
Planning & Requirement	🟩		
Design	🟩		
Coding & Implementation		🟧	
Testing & Report			🟦

Specific Requirements

Use Case Reports

1. **Administrator:** Responsible for managing student details.

Use-case: Login into the website

Goal in context: Gain access to the website

Brief Description: This use case is used when the administrator wants to access the website to enable/disable/update the personal details of the student.

Preconditions: The Administrator must be logged onto the website for this use case to begin.

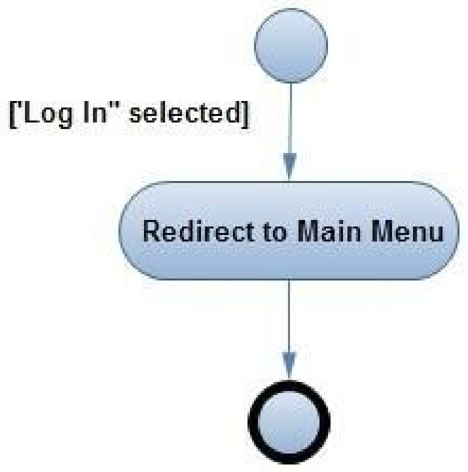
Basic Flow:

- ❖ The Website prompts the administrator for the username and password.
- ❖ The Administrator enters the username and password.
- ❖ The Website verifies the password and sets the user’s authorization.
- ❖ The Administrator is given access to the Website to perform his tasks

Alternative Flow:

- ❖ The administrator enters invalid username and password then he will not be allowed to enter the website**.**

Post conditions: The website state is unchanged by this use case.



Use Case Report- Login into the website

Use Case: Display student details

Goal in context: View the details of a student

Brief Description: This use case is used when the administrator wants to view the personal details of the students already existing in the database on the screen.

Preconditions:

- The Administrator must be logged into the system for this use case to begin
- The details of the student must pre-exist in the database
- The student id must be entered correctly.

Basic Flow:

- The Administrator logs onto the System.
- The Administrator search the student from following keys: -
- The System prompts for the student

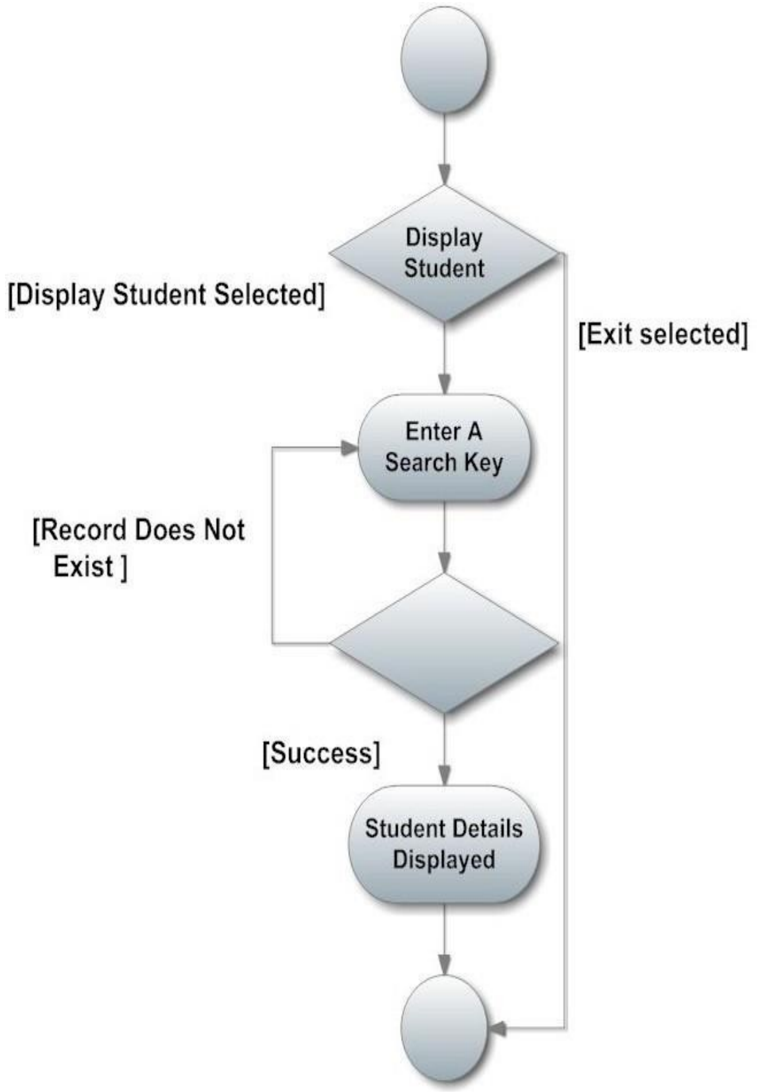
  detail from one of the above keys.
- The student details are displayed on

  the screen.

Alternative Flow:

Student Not Found

If in the **Display a student** sub-flows, a student with the specified id number does not exist, the system displays an error message. The Administrator can then enter a different id number or cancel the operation, at which point the use case ends.



Use Case Report-Display Student Details

Use Case: Edit student details

Goal in context: Edit the details of a student

Brief Description: This use case is used when the administrator wants to edit the personal details of himself/herself already existing in the database.

Preconditions:

- ❖ The Administrator must be logged into the system for this use case to begin.
- ❖ The details of the student must pre-exist in the database

Basic Flow:

The Administrator logs onto the System.

The Administrator can edit following keys: -

- ➢ First/last name
- ➢ Gender
- ➢ DOB
- ➢ Contact no Qualification
- ➢ City

- Email1
- Email2
- Address
- Description

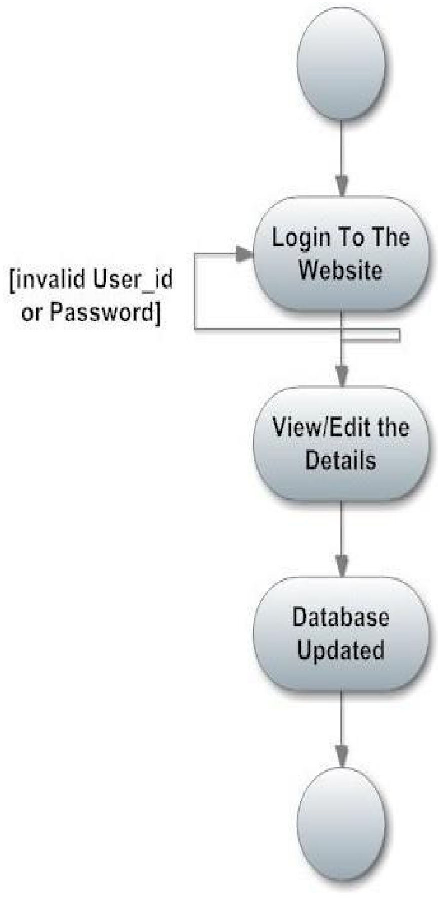
- The Website updates the database according to edited details.
- The student details are edited in the database.

Alternative Flow:

- There is no alternative flow of this use case diagram.

Post conditions:

- The student details get updated in the database**.**



Use Case Report- Edit student detail into thwebsite

2. Student

Use Case: student registration

Goal in context: Registration of a student

Brief Description: This use case is used when the student register himself/herself in the database online.

Preconditions:

- ❖ The student must access the website for this use case to begin.
- ❖ The user id must be unique and entered correctly.

Basic Flow:

The student enters the website.

The student fills his/her details from the following keys: -

- ➢ Student id
- ➢ password
- ➢ First/last name
- ➢ Status
- ➢ Gender
- ➢ DOB
- ➢ Contact no
- ➢ Qualification

- City
- Email1
- Email2
- Address
- Description
- Resume
- Image

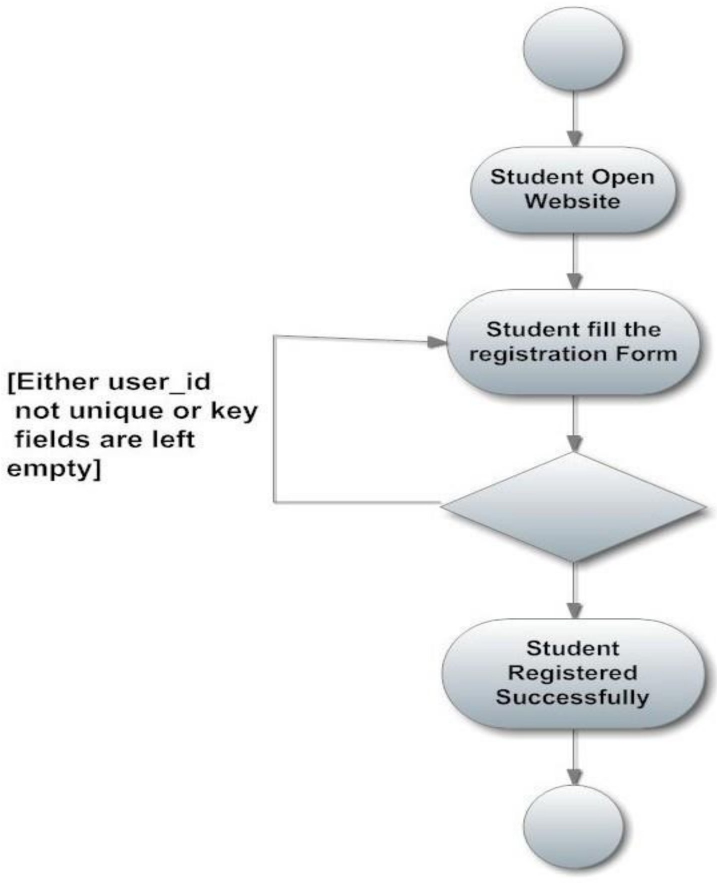
- The System details are added to the database.
- The student details are displayed on the screen.

Alternative Flow:

User ID not unique: if the user id entered is not unique then it will show an error message.

Post conditions:

The student gets registered on the website and to login into that particular the administrator must enable it.


Student Open Website
Student fill the registration Form
[Either user_id not unique or key fields are left empty]
Student Registered Successfully

Use Case Report- Register student on website

Use-case: Login into the website

Goal in context: Gain access to the website

Brief Description: This use case is used when the student wants to access the website

Preconditions: The Administrator must enable the student onto the website in order for this use case to begin.

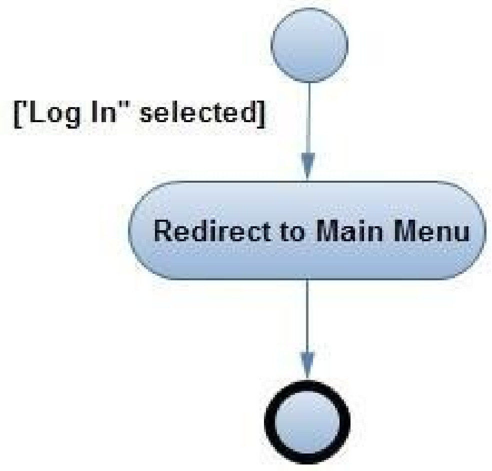
Basic Flow:

- The website prompts the student for the username and password.
- The student enters the username and password.
- The website verifies the password and sets the user's authorization.
- The student is given access to the website to perform his tasks.

Alternative Flow:

The student enters invalid username and password then he will not be allowed to enter the website**.**

Post conditions: The website state is unchanged by this use case.



Use Case Report- Login into the system

Use Case: Edit student details

Goal in context: Edit the details of a student

Brief Description: This use case is used when the student wants to edit the personal details of himself/herself already existing in the database.

Preconditions:

- The student must be logged into the system for this use case to begin.
- The details of the student must pre-exist in the database
- The student must be enabled by administrator.

Basic Flow:

- The student logs onto the System.
- The student can edit following keys: -

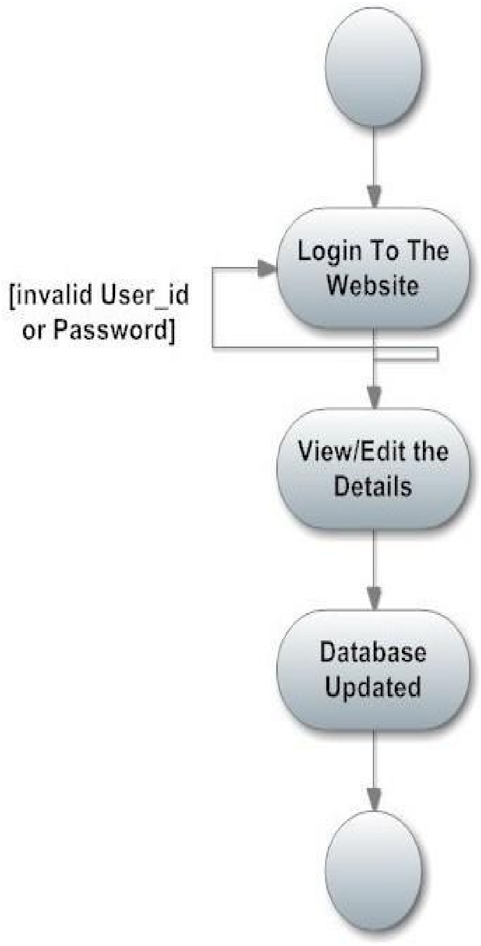
  - First/last name
  - Gender
  - DOB
  - Contact no
  - Qualification
  - City
  - Email1
  - Email2
  - Address
  - Description

- The Website updates the database according to edited details.
- The student details are edited in the database.

Alternative Flow: There is no alternative flow of this use case diagram.

Post conditions:

The student details get updated in the database**.**



Use Case Report- Edit Student Details into Database

SOURCE CODE

Login.js

```
import React, { useContext, useRef, useState } from "react";
import style from "./Login.module.css";
import { Link, Redirect, useHistory } from "react-router-dom";
import {userName , userPassword} from './auth';

const Login = (props) => {
  const emailRef = useRef();
  const passwordRef = useRef();
  const history = useHistory();
  const [error, setError] = useState("");
  const onLoginHandler = (e) => {
    e.preventDefault();
    const email = emailRef.current.value;
    const password = passwordRef.current.value;

      if (userName === email && userPassword === password) {
          props.onLoginHandler();
          history.push("/studentList")
      } else {
        setError("You've entered an invalid access credentials");
      }
  };

  return (
    <div className={style.login}>
      <h1>Admin Login</h1>
      <form onSubmit={onLoginHandler} className={style.form}>
        <div className={style.control}>
          <input type="email" id="email" ref={emailRef} placeholder="Email" />
        </div>
        <div className={style.control}>
          <input
            type="password"
            id="password"
            ref={passwordRef}
            placeholder="Password"
          />
        </div>
        {error !== "" && <p className={style.error}>{error}</p>}
        <div className={style.actions}>
          <button className="btn">Login</button>
        </div>
        <p>Forget your password?</p>
      </form>
    </div>
  );
};

export default Login;
```

Login.module.css

```
.login {
    width: 90%;
    max-width: 30rem;
    margin: 10rem auto;
    padding: 2rem;
  }

  .login h1 {
    text-align: center;
  }
```

```css
.control {
  margin: 1.5rem 0;
  display: flex;
  align-items: stretch;
  flex-direction: column;
}

.control input {
  flex: 3;
  font: inherit;
  padding: 0.8rem 0.6rem;
  border-radius: 6px;
  border: 1px solid #ccc;
}

.control input:focus {
  outline: none;
  border-color: #13a9d3;
  background: rgb(203, 235, 246);
}

.actions {
  display: flex;
  align-items: stretch;
  flex-direction: column;
  text-decoration: none;
}

.actionsLink {
  float: right;
}

.actions button {
  font: inherit;
  border-radius: 6px;
  padding: 0.8rem 1.5rem;
  border: 1px solid #8b005d;
  color: white;
  border-color: #21b4dd;
  background: #21b4dd;
  box-shadow: 0 0 4px rgba(0, 0, 0, 0.26);
  cursor: pointer;
  margin: 1rem 0;
}

.actions button:focus {
  outline: none;
}

.actions button:hover,
.actions button:active {
  background: #0fc0f1;
  box-shadow: 0 0 8px rgba(0, 0, 0, 0.26);
}

@media (min-width: 768px) {
  .control {
    align-items: center;
    flex-direction: row;
  }
}

.error {
  font: inherit;
  color: rgb(238, 16, 16);
}
```

About.js

```
import React from "react";

const About = () => {
  return (
    <div className='container' style={{ marginTop: "3rem", maxWidth: "50rem" }}>
      <div className='jumbotron'>
        <h2 style={{ color: "green", textAlign: "center" }}>About</h2>
        <p>
          This is a simple MERN stack web app to register a student with
          optional selection of majors.
        </p>
        <p>
      AMBRISH SHUKLA
        </p>
        <p>
```

```
          ABHINANDAN SAINI
        </p>
        <p>
          AADITYA SINGH
        </p>


      </div>
    </div>
  );
};

export default About;
```

Navbar.js

```
import React from "react";
import { Link, useHistory } from "react-router-dom";

import style from "./NavBar.module.css"
const Navbar = (props) => {
 const history = useHistory()
  const onLogoutHandler = () => {
    props.onLoginHandler();
    history.push("/");
  }

  return (
    <nav className='navbar navbar-dark bg-success navbar-expand-md'>
      <div className='container'>
        <Link
          to='/'
          className='navbar-brand'
          style={{ textTransform: "uppercase", fontSize: "30px" }}
        >
          Student Registration
        </Link>
        <div className='collapse navbar-collapse'>
```

```
          <ul className='navbar-nav ml-auto'>
            <li className='nav-item'>
              <Link to='/studentList' className='nav-link'>
                Students
              </Link>
            </li>
            <li className='nav-item'>
              <Link to='/majors' className='nav-link'>
                Majors
              </Link>
            </li>
            <li className='nav-item'>
              <Link to='/about' className='nav-link'>
                About
              </Link>
            </li>
            <li className='nav-item'>
              <button className={style.button} onClick={onLogoutHandler}>
                Logout
              </button>
            </li>
          </ul>
        </div>
      </div>
    </nav>
  );
};

export default Navbar;
```

NavBar.module.css

```
.button {
    outline: none;
    border: none;
    background-color: transparent;
    color: rgba(255, 255, 255, 0.6);
    margin-top: 7px;
}
```

Add Major

```
import React, { useState, useEffect } from "react";

import { useAppContext } from "../../context/AppContext";

// For Add and Edit Major Component
const AddMajor = ({ editableMajor, edit, setEdit }) => {
  const { addMajor, editMajor } = useAppContext();

  const [majorText, setMajorText] = useState("");

  useEffect(() => {
    // Fill input with majorText to be edited
    if (edit) setMajorText(editableMajor.major);
    // eslint-disable-next-line react-hooks/exhaustive-deps
  }, [edit]);

  const handleAddOrEditMajor = (e) => {
    e.preventDefault();

    if (!edit) {
      addMajor(majorText);
    } else {
      editMajor(editableMajor._id, majorText);
      setEdit(false);
    }

    setMajorText("");
  };
  return (
    <div>
      <h3 style={{ color: "green" }}>
        {edit ? "Update" : "Create"} a New Major
      </h3>
      <form onSubmit={handleAddOrEditMajor}>
        <div className='form-group'>
          <label>Major</label>
          <input
            type='text'
            required
            className='form-control'
            value={majorText}
            onChange={(e) => setMajorText(e.target.value)}
          />
        </div>
        <div className='d-flex'>
          <div className='form-group' style={{ marginRight: "0.5rem" }}>
            <input
              type='submit'
              value={`${edit ? "Update" : "Create"} Major`}
              className='btn btn-success'
            />
          </div>
          {edit && (
            <div className='form-group'>
              <button
                type='button'
                onClick={() => {
                  setEdit(false);
                  setMajorText("");
                }}
                className='btn btn-secondary'
              >
                Cancel Update
              </button>
            </div>
          )}
        </div>
      </form>
    </div>
  );
};
```

```
export default AddMajor;
```

MajorList.js

```
import React, { useState } from "react";

import AddMajor from "./AddMajor";
import { useAppContext } from "../../context/AppContext";

const MajorList = () => {
  const { majors, deleteMajor } = useAppContext();

  // For edit
  // Get major info from list and edit status
  const [editableMajor, setEditableMajor] = useState({});
  const [edit, setEdit] = useState(false);

  // Set major obj and status for editing
  const handleEdit = (major) => {
    setEditableMajor(major);
    setEdit(true);
  };

  const Major = ({ major }) => (
    <tr>
      <td>{major.major}</td>
      <td>
        <button
          className='btn btn-light btn-sm'
          onClick={() => handleEdit(major)}
        >
          edit
        </button>{" "}
        <button
          className='btn btn-light btn-sm'
          onClick={() => deleteMajor(major._id)}
        >
          delete
        </button>
      </td>
    </tr>
  );

  const majorList = () => {
    return majors.map((curMajor) => {
      return <Major major={curMajor} key={curMajor._id} />;
    });
  };

  return (
    <div className='container' style={{ marginTop: "3rem", maxWidth: "40rem" }}>
      <AddMajor editableMajor={editableMajor} edit={edit} setEdit={setEdit} />
      <hr />
      <h3 style={{ marginBottom: "2rem", color: "green" }}>List of Majors</h3>
      <table className='table table-success table-hover'>
        <thead className='thead-dark'>
          <tr>
            <th>Major</th>
            <th>Actions</th>
          </tr>
        </thead>
        <tbody>{majorList()}</tbody>
      </table>
    </div>
  );
};

export default MajorList;
```

AddStudent.js

```
import React, { useState } from "react";
import { Link, useHistory } from "react-router-dom";

import { useAppContext } from "../../context/AppContext";
```

```
const AddStudent = () => {
  const { majors, addStudent } = useAppContext();

  const history = useHistory();

  const [formData, setFormData] = useState({
    firstname: "",
    lastname: "",
    age: 0,
    major: "",
    email: "",
    phone: "",
  });

  const { firstname, lastname, age, major, email, phone } = formData;

  const handleChange = (e) => {
    setFormData({ ...formData, [e.target.name]: e.target.value });
  };

  const handleAddStudent = (e) => {
    e.preventDefault();

    addStudent(formData);

    setFormData({
      firstname: "",
      lastname: "",
      age: 0,
      major: "",
      email: "",
      phone: "",
    });

    history.push("/");
  };

  return (
    <div className='container' style={{ marginTop: "3rem", maxWidth: "40rem" }}>
      <h3 style={{ marginTop: "3rem", marginBottom: "2rem", color: "green" }}>
        Create a New Student
      </h3>
```

```
<form onSubmit={handleAddStudent}>
  <div className='form-group'>
    <label>First Name</label>
    <input
      type='text'
      required
      className='form-control'
      name='firstname'
      value={firstname}
      onChange={handleChange}
    />
  </div>

  <div className='form-group'>
    <label>Last Name</label>
    <input
      type='text'
      required
      className='form-control'
      name='lastname'
      value={lastname}
      onChange={handleChange}
    />
  </div>

  <div className='form-group'>
    <label>Age</label>
    <input
      type='number'
      required
      className='form-control'
      name='age'
      value={age}
      onChange={handleChange}
    />
  </div>

  <div className='form-group'>
    <label>Major</label>
    <select
      required
      className='form-control'
      name='major'
      value={major}
      onChange={handleChange}
    >
      <option value='' disabled>
        Selec a major
      </option>
      {majors.map((item) => (
        <option value={item._id} key={item._id}>
          {item.major}
        </option>
      ))}
    </select>
  </div>

  <div className='form-group'>
    <label>Email</label>
    <input
      type='email'
      required
      className='form-control'
      name='email'
      value={email}
      onChange={handleChange}
    />
  </div>

  <div className='form-group'>
    <label>Phone</label>
    <input
      type='text'
      required
      className='form-control'
      name='phone'
      value={phone}
      onChange={handleChange}
    />
  </div>

  <div className='form-group'>
    <input
      type='submit'
      value='Create Student'
      className='btn btn-success'
```

```
          />{" "}
          <Link to='/' className='btn btn-secondary'>
            Cancel
          </Link>
        </div>
      </form>
    </div>
  );
};

export default AddStudent;
```

EditStudent.js

```
import React, { useState } from "react";
import { Link, useHistory } from "react-router-dom";

import { useAppContext } from "../../context/AppContext";

const EditStudent = () => {
  const { editStudent, student, majors } = useAppContext();

  const history = useHistory();

  const [formData, setFormData] = useState({
    firstname: student.firstname,
    lastname: student.lastname,
    age: student.age,
    major: student.major._id,
    email: student.email,
    phone: student.phone,
  });

  const { firstname, lastname, age, major, email, phone } = formData;
  const handleChange = (e) => {
    setFormData({ ...formData, [e.target.name]: e.target.value });
  };

  const handleEditStudent = (e) => {
    e.preventDefault();

    editStudent(student._id, formData);

    setFormData({
      firstname: "",
      lastname: "",
      age: 0,
      major: "",
      email: "",
      phone: "",
    });

    history.push("/");
  };

  return (
    <div className='container' style={{ marginTop: "3rem", maxWidth: "40rem" }}>
      <h3 style={{ color: "green", marginBottom: "2rem", marginTop: "3rem" }}>
        Update Student
      </h3>
      <form onSubmit={handleEditStudent}>
        <div className='form-group'>
          <label>First Name</label>
          <input
            type='text'
            required
            className='form-control'
            name='firstname'
            value={firstname}
            onChange={handleChange}
          />
        </div>

        <div className='form-group'>
          <label>Last Name</label>
          <input
            type='text'
            required
            className='form-control'
            name='lastname'
            value={lastname}
            onChange={handleChange}
          />
        </div>

        <div className='form-group'>
          <label>Age</label>
          <input
            type='number'
            required
            className='form-control'
            name='age'
            value={age}
            onChange={handleChange}
          />
        </div>

        <div className='form-group'>
          <label>Major</label>
          <select
            required
            className='form-control'
            name='major'
            value={major}
            onChange={handleChange}
          >
            <option value='' disabled>
              Select a major
            </option>
            {majors.map((item) => (
              <option value={item._id} key={item._id}>
                {item.major}
              </option>
            ))}
          </select>
        </div>

        <div className='form-group'>
          <label>Email</label>
          <input
```

```
            type='email'
            required
            className='form-control'
            name='email'
            value={email}
            onChange={handleChange}
          />
        </div>

        <div className='form-group'>
          <label>Phone</label>
          <input
            type='text'
            required
            className='form-control'
            name='phone'
            value={phone}
            onChange={handleChange}
          />
        </div>

        <div className='form-group'>
          <input
            type='submit'
            value='Update Student'
            className='btn btn-success'
          />{" "}
          <Link to='/' className='btn btn-secondary'>
            Cancel
          </Link>
        </div>
      </form>
    </div>
  );
};

export default EditStudent;
```

StudentDetails.js

```
import React, { useEffect } from "react";
import { Link, useParams } from "react-router-dom";

import { useAppContext } from "../../context/AppContext";

const StudentDetails = () => {
  const { getStudent, student, loading } = useAppContext();

  const { id } = useParams();

  useEffect(() => {
    getStudent(id);
    // eslint-disable-next-line react-hooks/exhaustive-deps
  }, []);

  const { _id, firstname, lastname, age, major, email, phone } = student;

  if (!student || loading) return <h2>Loading</h2>;

  return (
    <div className='container' style={{ marginTop: "3rem", maxWidth: "40rem" }}>
      <h3 style={{ marginTop: "3rem", color: "green" }}>Student Details</h3>
      <div className='card' style={{ width: "30rem", marginTop: "2rem" }}>
        <div className='card-body'>
          <h4
            className='card-title'
            style={{ color: "brown", textTransform: "uppercase" }}
          >
            {firstname} {lastname}
          </h4>
          <hr />
          <p className='card-text'>
            <strong>Age:</strong> {age}
          </p>
          <p className='card-text'>
            <strong>Major:</strong> {major && major.major}
          </p>
          <p className='card-text'>
            <strong>Email:</strong> {email}
          </p>
          <p className='card-text'>
            <strong>Phone:</strong> {phone}
          </p>
          <hr />
          <Link
            to={`/students/edit/${_id}`}
            style={{ paddingLeft: "1.5rem", paddingRight: "1.5rem" }}
            className='btn btn-success'
          >
            Edit
          </Link>{" "}
          <Link
            to='/'
            style={{ paddingLeft: "1.5rem", paddingRight: "1.5rem" }}
            className='btn btn-secondary'
          >
            Back
          </Link>
        </div>
      </div>
    </div>
  );
};

export default StudentDetails;
```

StudentItem.js

```
import React from "react";
import { Link } from "react-router-dom";

import { useAppContext } from "../../context/AppContext";

const StudentItem = ({ student }) => {
  const { deleteStudent } = useAppContext();
  return (
    <tr>
      <td>{student.firstname}</td>
      <td>{student.lastname}</td>
      <td>{student.major.major}</td>
      <td>
        <Link className='btn btn-light btn-sm' to={`/students/${student._id}`}>
          view
        </Link>{" "}
        <button
          className='btn btn-light btn-sm'
          onClick={() => deleteStudent(student._id)}
        >
          delete
        </button>
      </td>
    </tr>
  );
};

export default StudentItem;
```

StudentList.js

```
import React, { useEffect } from "react";
import { Link } from "react-router-dom";

import { useAppContext } from "../../context/AppContext";
import StudentItem from "./StudentItem";

const StudentList = () => {
  const { students, getStudents } = useAppContext();

  useEffect(() => {
    getStudents();
    // eslint-disable-next-line react-hooks/exhaustive-deps
  }, []);

  const studentList = () => {
    return students.map((curStudent) => {
      return <StudentItem student={curStudent} key={curStudent._id} />;
    });
  };
  return (
    <div style={{ marginTop: "3rem" }}>
      <h3 style={{ marginBottom: "2rem", color: "green" }}>List of Students</h3>
      <Link
        to='/students/add'
        className='btn btn-success'
        style={{ marginBottom: "2rem" }}
      >
        Create a New Student
      </Link>
      <table className='table  table-hover table-success'>
        <thead className='thead-dark'>
          <tr>
            <th>First Name</th>
            <th>Last Name</th>
            <th>Major</th>
            <th>Actions</th>
          </tr>
        </thead>
        <tbody>{studentList()}</tbody>
      </table>
    </div>
  );
};

export default StudentList;
```

ActionTypes.js

```
export const STUDENTS_GET = "STUDENTS_GET";
export const STUDENT_GET = "STUDENT_GET";
export const STUDENT_ADD = "STUDENT_ADD";
export const STUDENT_EDIT = "STUDENT_EDIT";
export const STUDENT_DELETE = "STUDENT_DELETE";
export const STUDENT_ERROR = "STUDENT_ERROR";
export const MAJORS_GET = "MAJORS_GET";
export const MAJOR_ADD = "MAJOR_ADD";
export const MAJOR_EDIT = "MAJOR_EDIT";
export const MAJOR_DELETE = "MAJOR_DELETE";
export const MAJOR_ERROR = "MAJOR_ERROR";
```

AppContext.js

```
import React, { useReducer, createContext, useContext } from "react";
import axios from "axios";

import reducer from "./AppReducer";
import {
  STUDENTS_GET,
  STUDENT_GET,
  STUDENT_ADD,
  STUDENT_EDIT,
  STUDENT_DELETE,
  STUDENT_ERROR,
  MAJORS_GET,
  MAJOR_ADD,
  MAJOR_EDIT,
  MAJOR_DELETE,
  MAJOR_ERROR,
} from "./ActionTypes";
```

```
const AppContext = createContext();

const initialState = {
  students: [],
  student: {},
  majors: [],
  major: {},
  loading: true,
  error: {},
};

const AppProvider = ({ children }) => {
  const [state, dispatch] = useReducer(reducer, initialState);

  // Get all majors
  const getMajors = async () => {
    try {
      const { data } = await axios.get("/api/majors");

      dispatch({ type: MAJORS_GET, payload: data });
    } catch (err) {
      dispatch({ type: MAJOR_ERROR, payload: err.message.error });
    }
  };

  // Add a major
  const addMajor = async (majorText) => {
    try {
      const config = {
        "Content-Type": "application/json",
      };
      const { data } = await axios.post(
        "/api/majors/create",
        { majorText },
        config
      );

      dispatch({ type: MAJOR_ADD, payload: data });
    } catch (err) {
      dispatch({ type: MAJOR_ERROR, payload: err.response.data.error[0] });
    }
  };
```

```
// Edit major
const editMajor = async (majId, majorText) => {
  try {
    const config = {
      "Content-Type": "application/json",
    };
    const { data } = await axios.put(
      `/api/majors/update/${majId}`,
      { majorText },
      config
    );

    dispatch({ type: MAJOR_EDIT, payload: { id: majId, major: data } });
  } catch (err) {
    dispatch({ type: MAJOR_ERROR, payload: err.response.data.error[0] });
  }
};

// Delete major
const deleteMajor = async (majId) => {
  try {
    await axios.delete(`/api/majors/${majId}`);

    dispatch({ type: MAJOR_DELETE, payload: majId });
  } catch (err) {
    dispatch({ type: MAJOR_ERROR, payload: err.response.data.error[0] });
  }
};

// Get all students
const getStudents = async () => {
  try {
    const { data } = await axios.get("/api/students");

    dispatch({ type: STUDENTS_GET, payload: data });
  } catch (err) {
    console.log(err.message);
    dispatch({ type: STUDENT_ERROR, payload: err.message.error });
  }
};

// Get a single student
const getStudent = async (stuId) => {
  try {
    const { data } = await axios.get(`/api/students/${stuId}`);

    dispatch({ type: STUDENT_GET, payload: data });
  } catch (err) {
    dispatch({ type: STUDENT_ERROR, payload: err.response.data.error[0] });
  }
};

// Add a student
const addStudent = async (formData) => {
  try {
    const config = {
      "Content-Type": "application/json",
    };
    const { data } = await axios.post(
      "/api/students/create",
      formData,
      config
    );

    dispatch({ type: STUDENT_ADD, payload: data });
  } catch (err) {
    dispatch({ type: STUDENT_ERROR, payload: err.response.data.error[0] });
  }
};

// Edit a student
const editStudent = async (stuId, formData) => {
  try {
    const config = {
      "Content-Type": "application/json",
    };
    const { data } = await axios.put(
      `/api/students/update/${stuId}`,
      formData,
      config
    );

    dispatch({
      type: STUDENT_EDIT,
      payload: { id: stuId, updatedStudent: data },
    });
    } catch (err) {
      dispatch({ type: STUDENT_ERROR, payload: err.response.data.error[0] });
    }
  };

  // Delete student
  const deleteStudent = async (stuId) => {
    try {
      await axios.delete(`/api/students/${stuId}`);

      dispatch({ type: STUDENT_DELETE, payload: stuId });
    } catch (err) {
      dispatch({ type: STUDENT_ERROR, payload: err.response.data.error[0] });
    }
  };

  return (
    <AppContext.Provider
      value={{
        ...state,
        getMajors,
        addMajor,
        editMajor,
        deleteMajor,
        getStudents,
        getStudent,
        addStudent,
        editStudent,
        deleteStudent,
      }}
    >
      {children}
    </AppContext.Provider>
  );
};

const useAppContext = () => {
  return useContext(AppContext);
};

export { AppProvider, useAppContext };
```

AppReducer.js

```
import {
  STUDENTS_GET,
  STUDENT_GET,
  STUDENT_ADD,
  STUDENT_EDIT,
  STUDENT_DELETE,
  STUDENT_ERROR,
  MAJORS_GET,
  MAJOR_ADD,
  MAJOR_EDIT,
  MAJOR_DELETE,
  MAJOR_ERROR,
} from "./ActionTypes";

const reducer = (state, action) => {
  const { type, payload } = action;

  switch (type) {
    case MAJORS_GET:
      return { ...state, loading: false, majors: payload };
    case MAJOR_ADD:
      return { ...state, loading: false, majors: [payload, ...state.majors] };
    case MAJOR_EDIT:
      return {
        ...state,
        loading: false,
        // Replace the edit major with payload in the list
        majors: state.majors.map((maj) =>
          maj._id === payload.id ? { ...payload.major } : maj
        ),
      };
    case MAJOR_DELETE:
      return {
        ...state,
        loading: false,
        majors: state.majors.filter((maj) => maj._id !== payload),
      };
    case STUDENTS_GET:
      return { ...state, loading: false, students: payload };
    case STUDENT_GET:
      return { ...state, loading: false, student: payload };
    case STUDENT_ADD:
      return {
        ...state,
        loading: false,
        students: [payload, ...state.students],
      };
    case STUDENT_EDIT:
      return {
        ...state,
        loading: false,
        // Replace the edit student with payload in the list
        students: state.students.map((stu) =>
          stu._id === payload.id ? { ...payload.updatedStudent } : stu
        ),
      };
    case STUDENT_DELETE:
      return {
        ...state,
        loading: false,
        students: state.students.filter((stu) => stu._id !== payload),
      };
    case MAJOR_ERROR:
      return { ...state, loading: false, error: payload };
    case STUDENT_ERROR:
      return { ...state, loading: false, error: payload };
    default:
      return state;
  }
};

export default reducer;
```

App.css

```
.background {
    background: url("./assets//screenshot/student.jpg") center no-repeat ;
    /* opacity: 0.8; */
    height: 30rem;
    margin-top: 2rem;
}

.container {
    z-index: 100;
}
```

App.js

```
import React, { useEffect, useState } from "react";
import { BrowserRouter, Switch, Route } from "react-router-dom";

import "bootstrap/dist/css/bootstrap.min.css";

import { useAppContext } from "./context/AppContext";

import Navbar from "./components/layout/Navbar";
import About from "./components/layout/About";
import StudentList from "./components/student/StudentList";
import StudentDetails from "./components/student/StudentDetails";
import AddStudent from "./components/student/AddStudent";
import EditStudent from "./components/student/EditStudent";
import MajorList from "./components/major/MajorList";
import AddMajor from "./components/major/AddMajor";
import Login from "./components/Authentication/Login";
import './App.css'

const App = () => {
  const { getStudents, getMajors } = useAppContext();

  const [loginStatus, setLoginStatus] = useState(false);

  const onLoginHandler = () => {
    setLoginStatus(!loginStatus)
  }

  useEffect(() => {
    getStudents();
    getMajors();
    // eslint-disable-next-line react-hooks/exhaustive-deps
  }, []);

  return (

    <BrowserRouter>
      {loginStatus && <Navbar onLoginHandler={onLoginHandler}/>}
      {/* <div className=' background' /> */}
      <div className=' container'>
        <Switch>
          {!loginStatus && <Route exact path="/"><Login
onLoginHandler={onLoginHandler}/></Route>}
          <Route exact path='/studentList'> <StudentList /> </Route>
          <Route exact path='/about' component={About} />
          <Route exact path='/students/add' component={AddStudent} />
          <Route exact path='/students/edit/:id' component={EditStudent} />
          <Route exact path='/students/:id' component={StudentDetails} />
          <Route exact path='/majors' component={MajorList} />
          <Route exact path='/majors/add' component={AddMajor} />
        </Switch>
      </div>
    </BrowserRouter>
  );
};

export default App;
```

Index.css

```
body {
  margin: 0;
  font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI', 'Roboto',
'Oxygen',
    'Ubuntu', 'Cantarell', 'Fira Sans', 'Droid Sans', 'Helvetica Neue',
    sans-serif;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;

}

code {
  font-family: source-code-pro, Menlo, Monaco, Consolas, 'Courier New',
    monospace;
}
```

Index.js

```
import React  from  "react";
import ReactDOM from "react-dom";
import "./index.css";
import App from "./App";

import { AppProvider } from "./context/AppContext";

ReactDOM.render(
  <AppProvider>
    <App />
  </AppProvider>,
  document.getElementById("root")
);
```

Major.js

```
const mongoose = require("mongoose");

const majorSchema = mongoose.Schema(
  {
    major: {
      type: String,
      required: true,
      unique: true,
      trim: true,
      minlength: 3,
    },
  },
  { timestamps: true }
);

module.exports = mongoose.model("Major", majorSchema);
```

Student.js

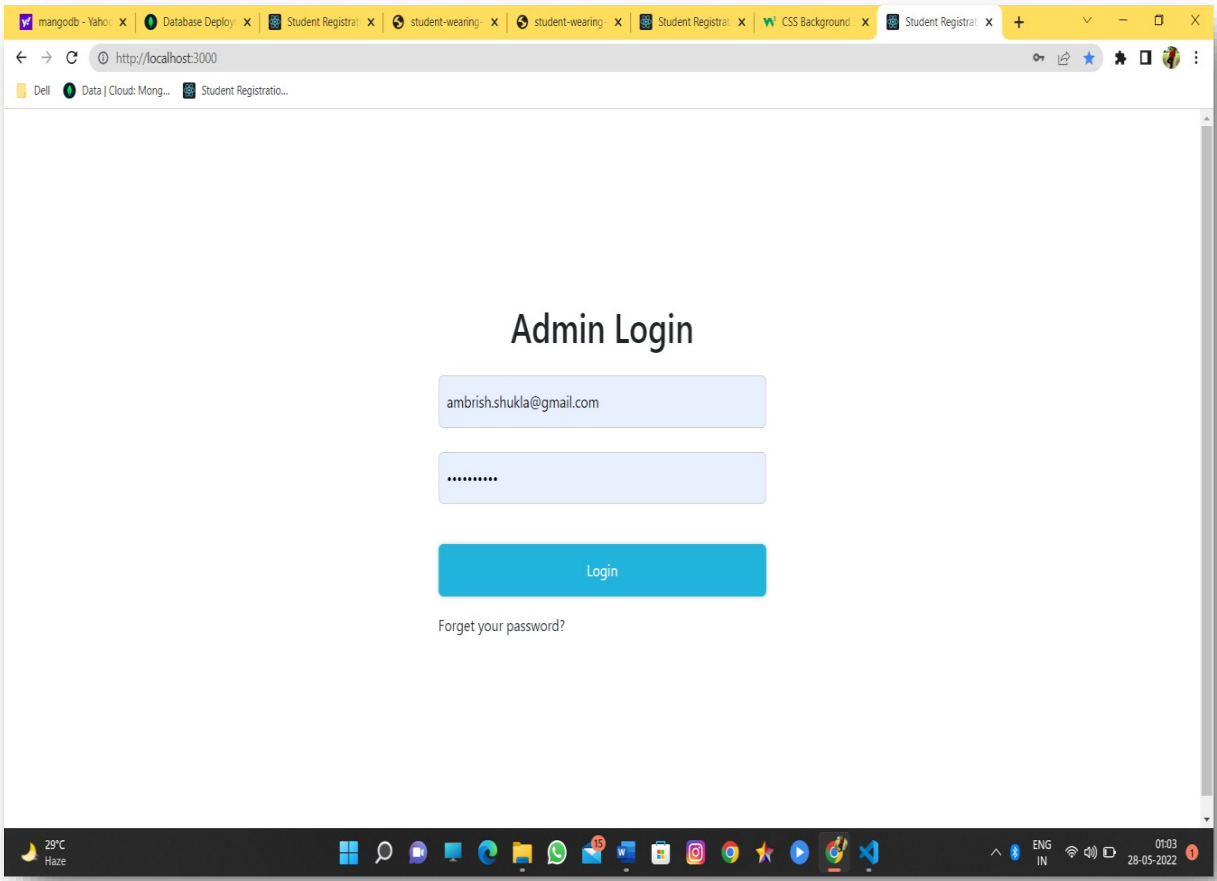
```
const mongoose = require("mongoose");

const studentSchema = mongoose.Schema(
  {
    firstname: { type: String, required: true },
    lastname: { type: String, required: true },
    age: { type: Number, required: true },
    major: { type: mongoose.Schema.Types.ObjectId, ref: "Major" },
    email: { type: String, required: true },
    phone: { type: String, required: true },
  },
  { timestamps: true }
);

module.exports = mongoose.model("Student", studentSchema);
```

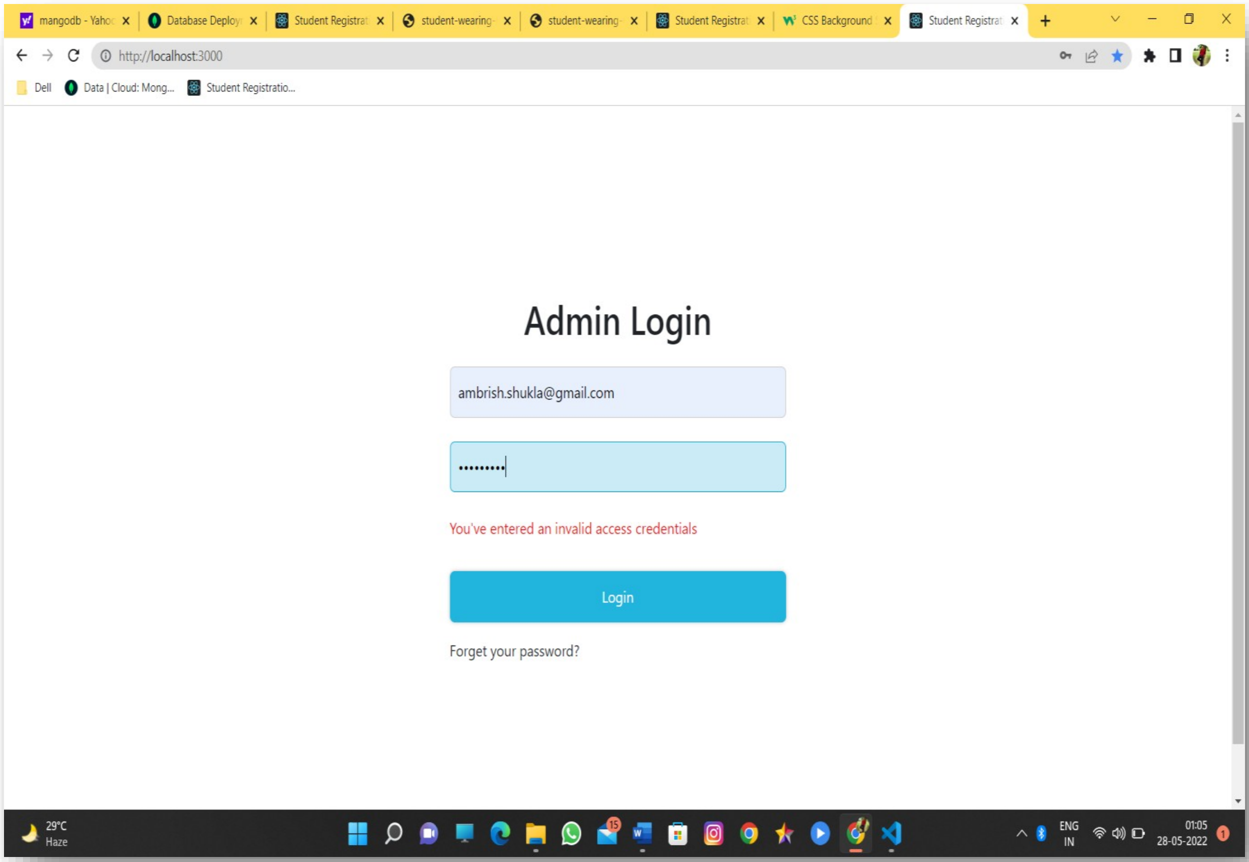
Server.js

```
const express = require("express");
const mongoose = require("mongoose");
const cors = require("cors");
const dotenv = require("dotenv");
const morgan = require("morgan");
const path = require("path");

dotenv.config();

const app = express();

// Middleware
app.use(express.json());
app.use(cors());

if (process.env.NODE_ENV === "development") {
  app.use(morgan("dev"));
}

// MongoDB
mongoose.connect(process.env.MONGO_URI, {
  useCreateIndex: true,
  useNewUrlParser: true,
  useUnifiedTopology: true,
  useFindAndModify: false,
});

const connection = mongoose.connection;

connection.once("open", () =>
  console.log("MongoDB Database Connection Established Successfully!!")
);

// First route - Welcome on Heroku
// app.get("/", (req, res) => res.send("API Running"));

// routes
const studentRoutes = require("./routes/students");
const majorRoutes = require("./routes/majors");
```

```
app.use("/api/students", studentRoutes);
app.use("/api/majors", majorRoutes);

// DEPLOY - Serve static assets if in production
if (process.env.NODE_ENV === "production") {
  // Set static folder
  app.use(express.static("client/build"));

  app.get("*", (req, res) => {
    res.sendFile(path.resolve(__dirname, "client", "build", "index.html"));
  });
}

// port
const PORT = process.env.PORT || 5000;

app.listen(PORT, () => console.log(`Server is running in PORT ${PORT}`));
```

SNAPSHOTS


http://localhost:3000
Admin Login
ambrish.shukla@gmail.com
••••••••••
Login
Forget your password?

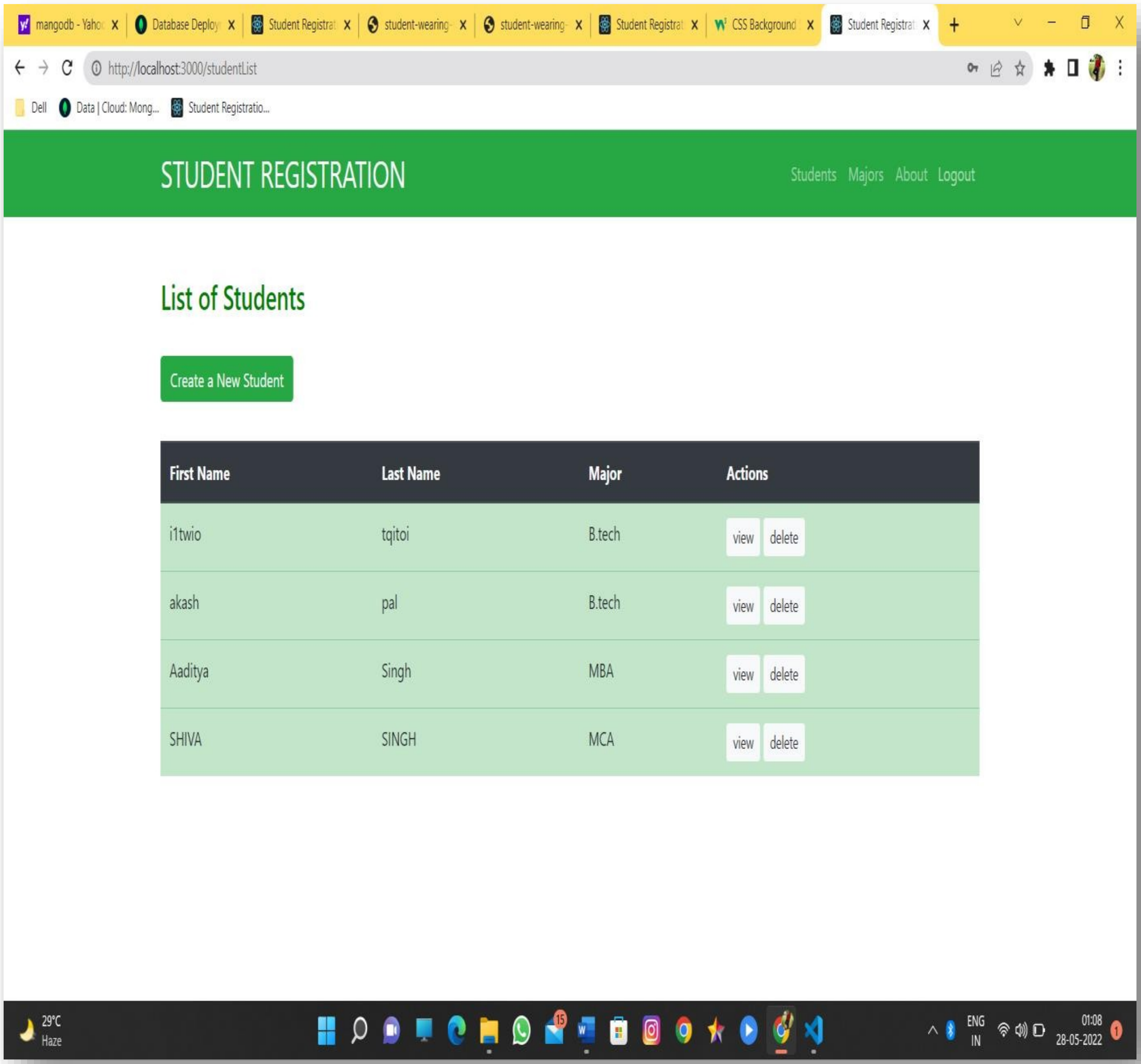


Admin Login

ambrish.shukla@gmail.com

•••••••••

You've entered an invalid access credentials



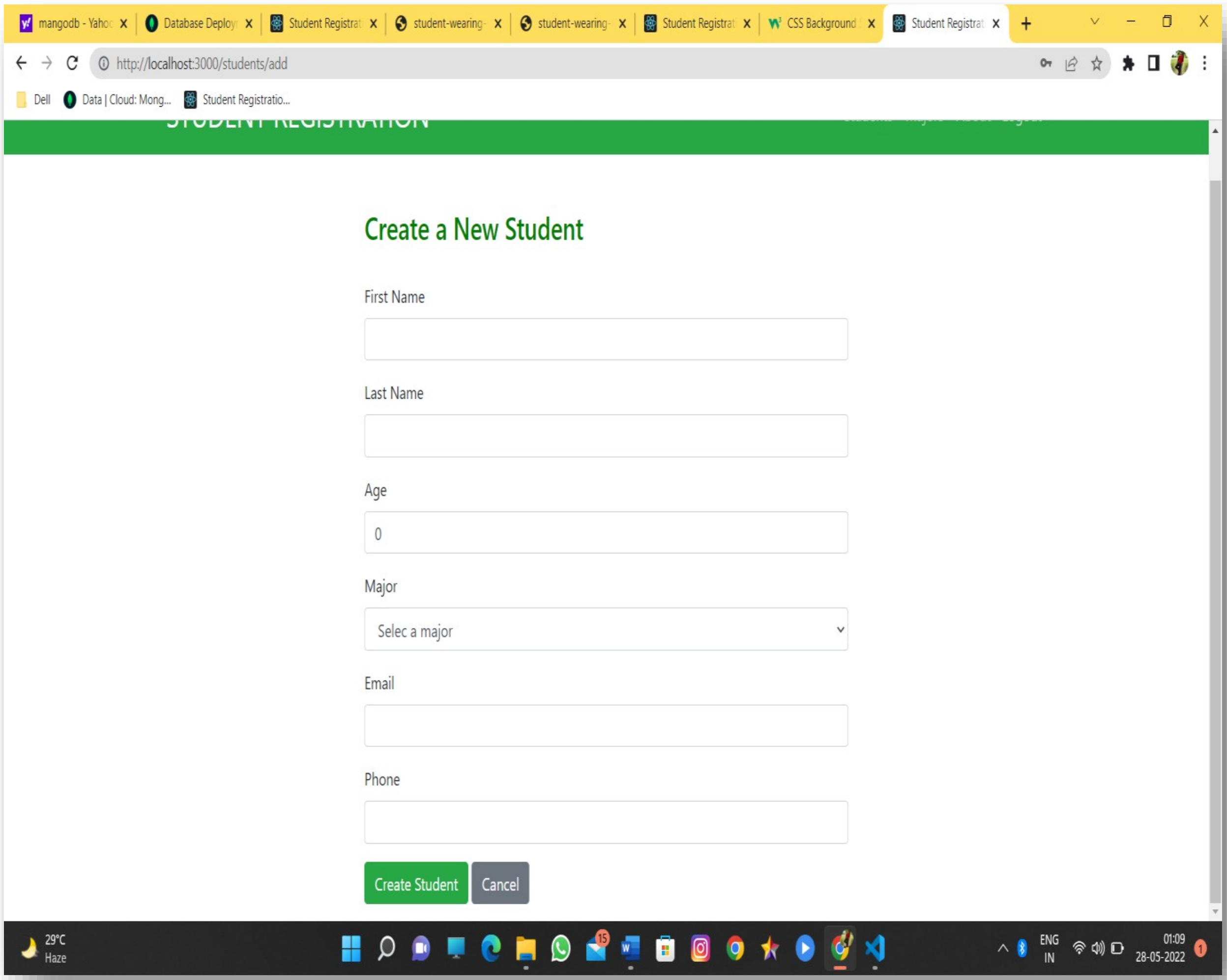
STUDENT REGISTRATION

Students Majors About Logout

List of Students

Create a New Student

First Name	Last Name	Major	Actions
i1twio	tqitoi	B.tech	view delete
akash	pal	B.tech	view delete
Aaditya	Singh	MBA	view delete
SHIVA	SINGH	MCA	view delete


http://localhost:3000/students/add

Create a New Student

First Name

Last Name

Age

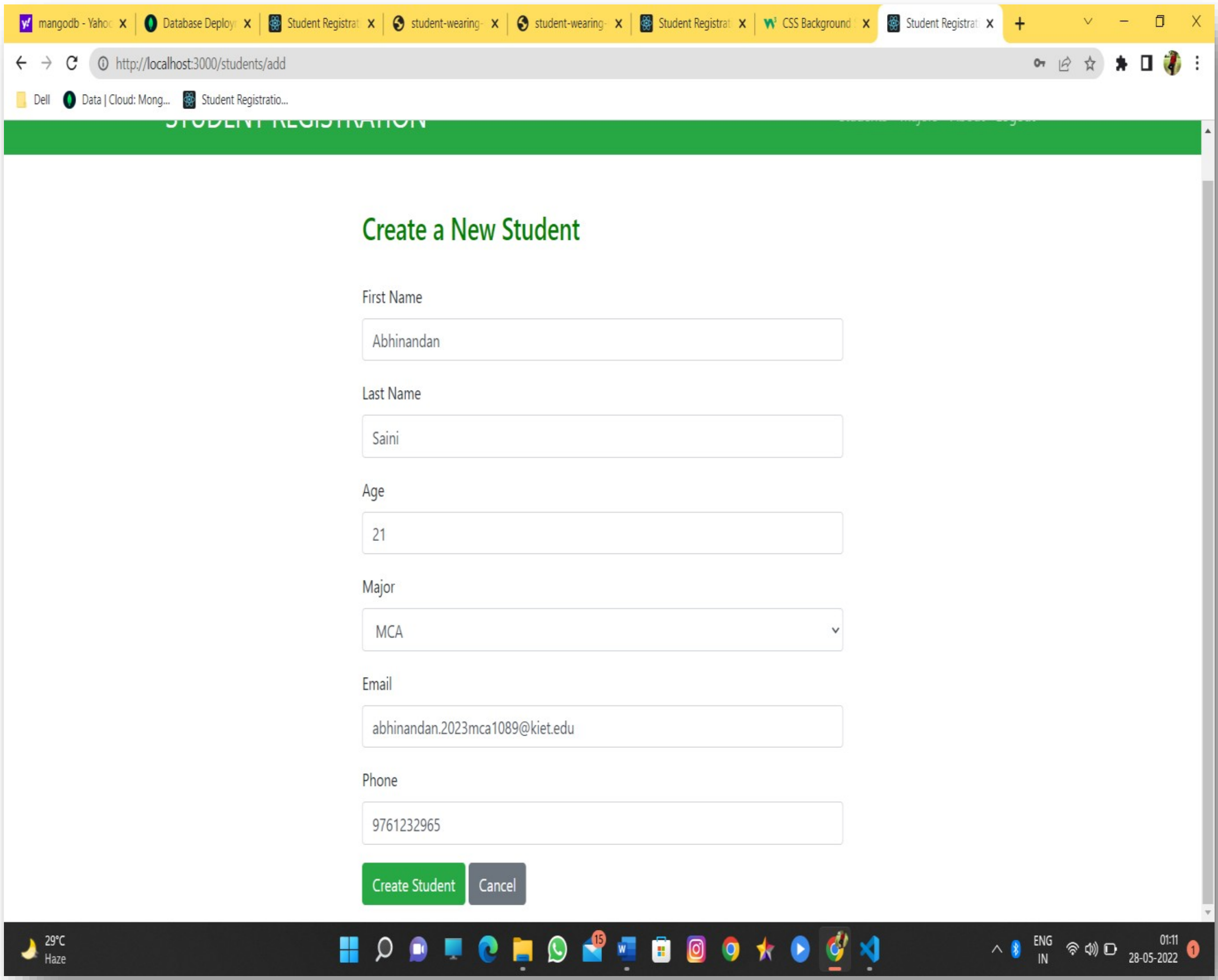
0

Major

Selec a major

Email

Phone



Create a New Student

First Name
Abhinandan

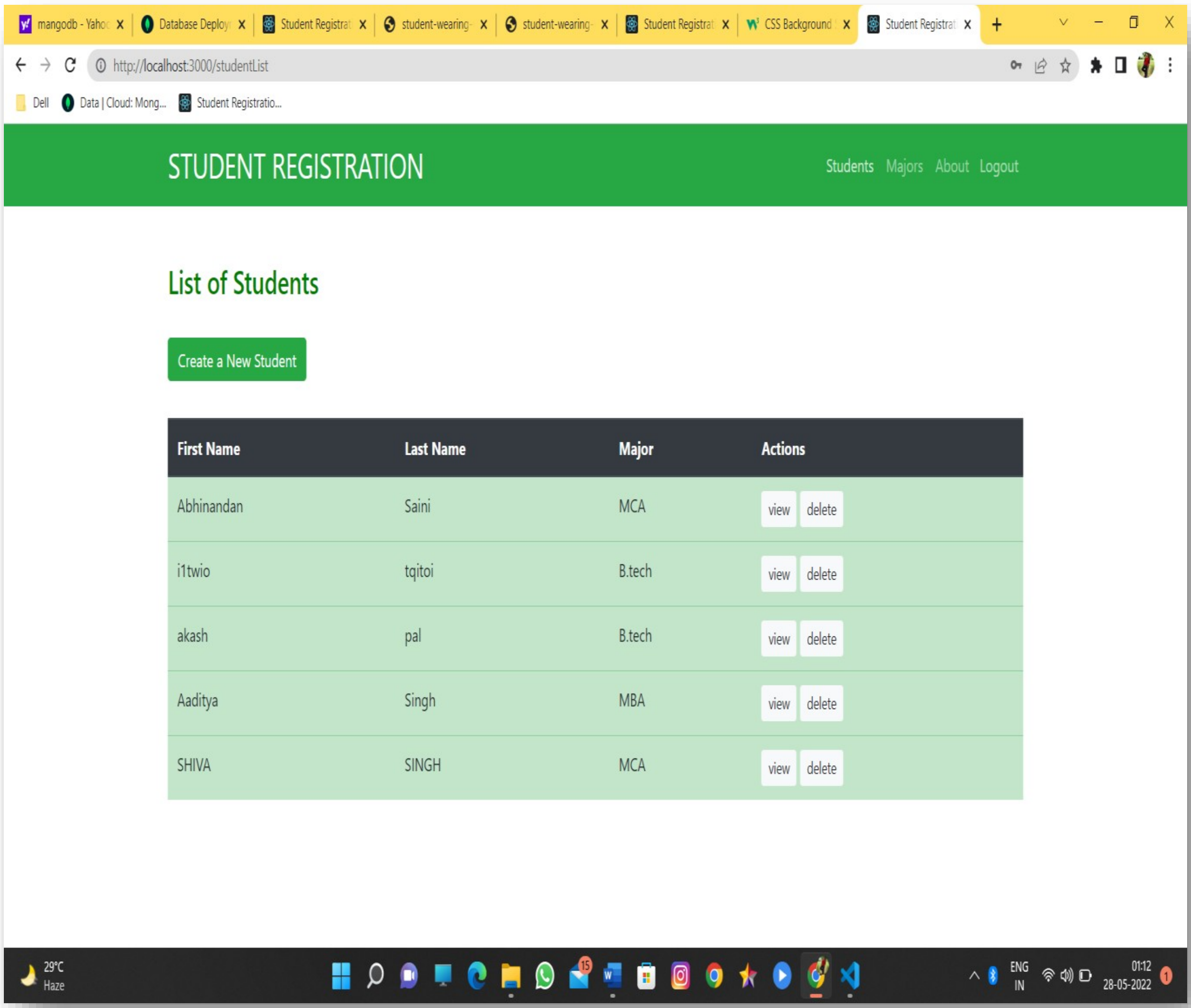
Last Name
Saini

Age
21

Major
MCA

Email
abhinandan.2023mca1089@kiet.edu

Phone
9761232965

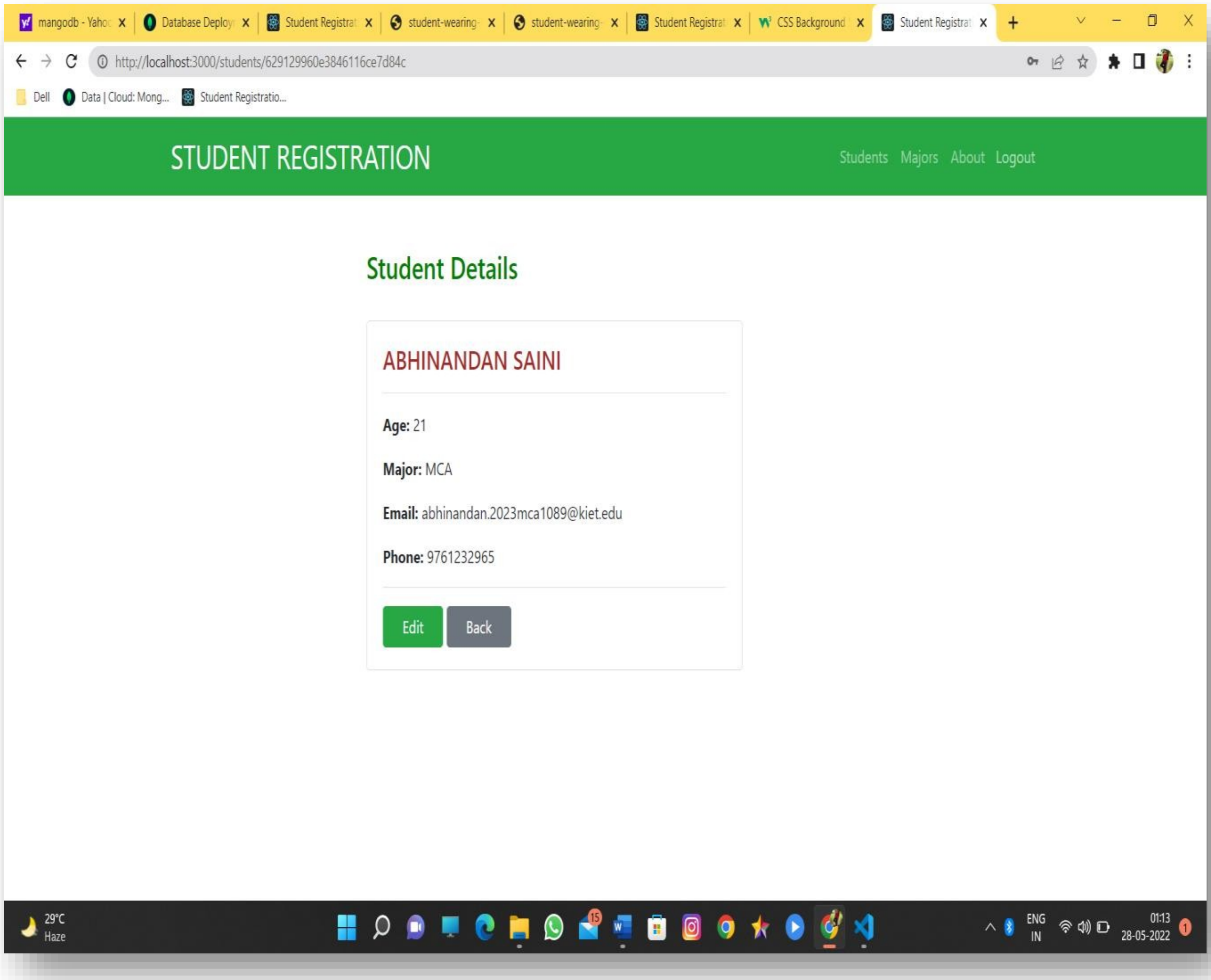


STUDENT REGISTRATION

List of Students

Create a New Student

First Name	Last Name	Major	Actions
Abhinandan	Saini	MCA	view delete
i1twio	tqitoi	B.tech	view delete
akash	pal	B.tech	view delete
Aaditya	Singh	MBA	view delete
SHIVA	SINGH	MCA	view delete



STUDENT REGISTRATION

Student Details

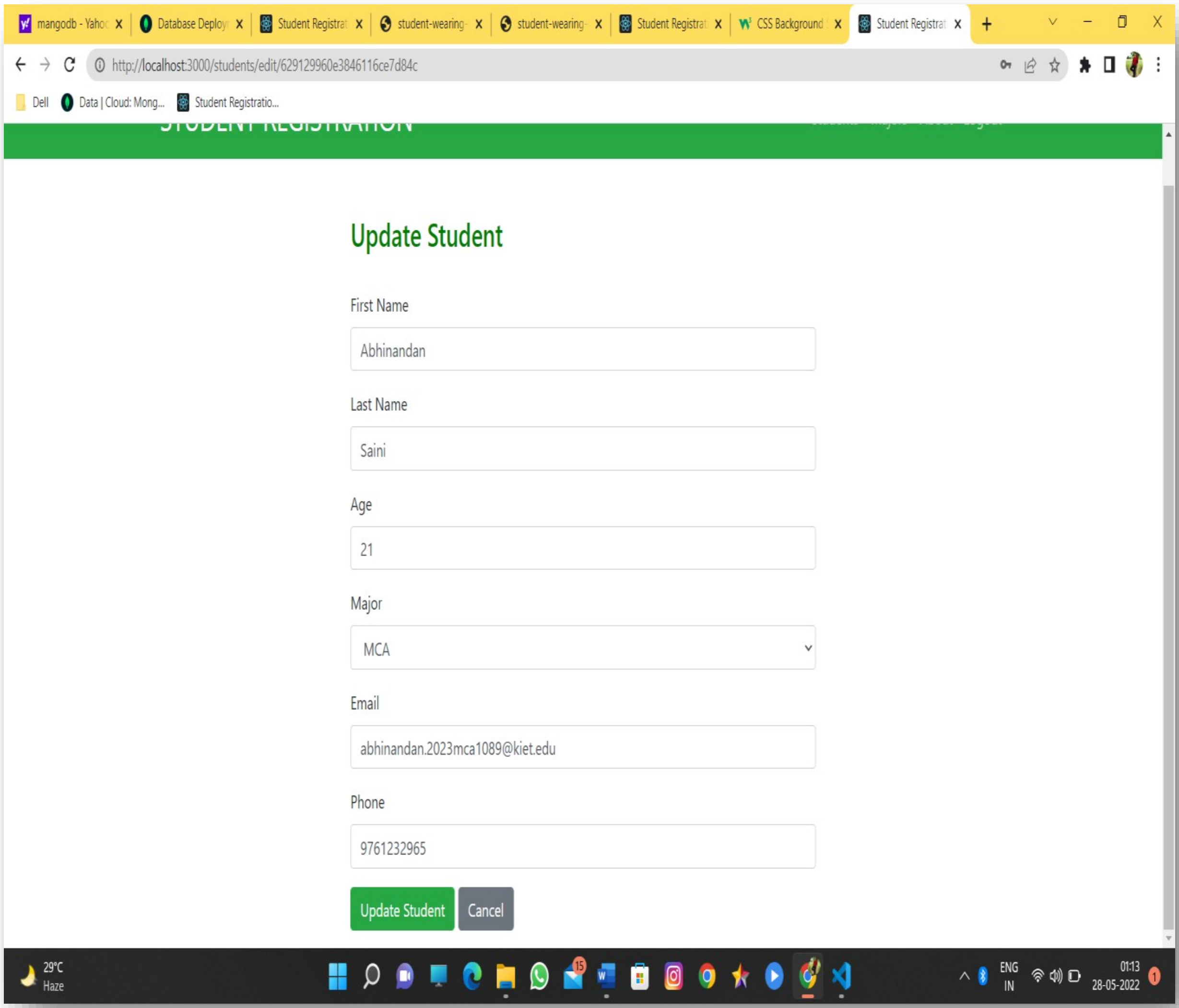
ABHINANDAN SAINI

Age: 21

Major: MCA

Email: abhinandan.2023mca1089@kiet.edu

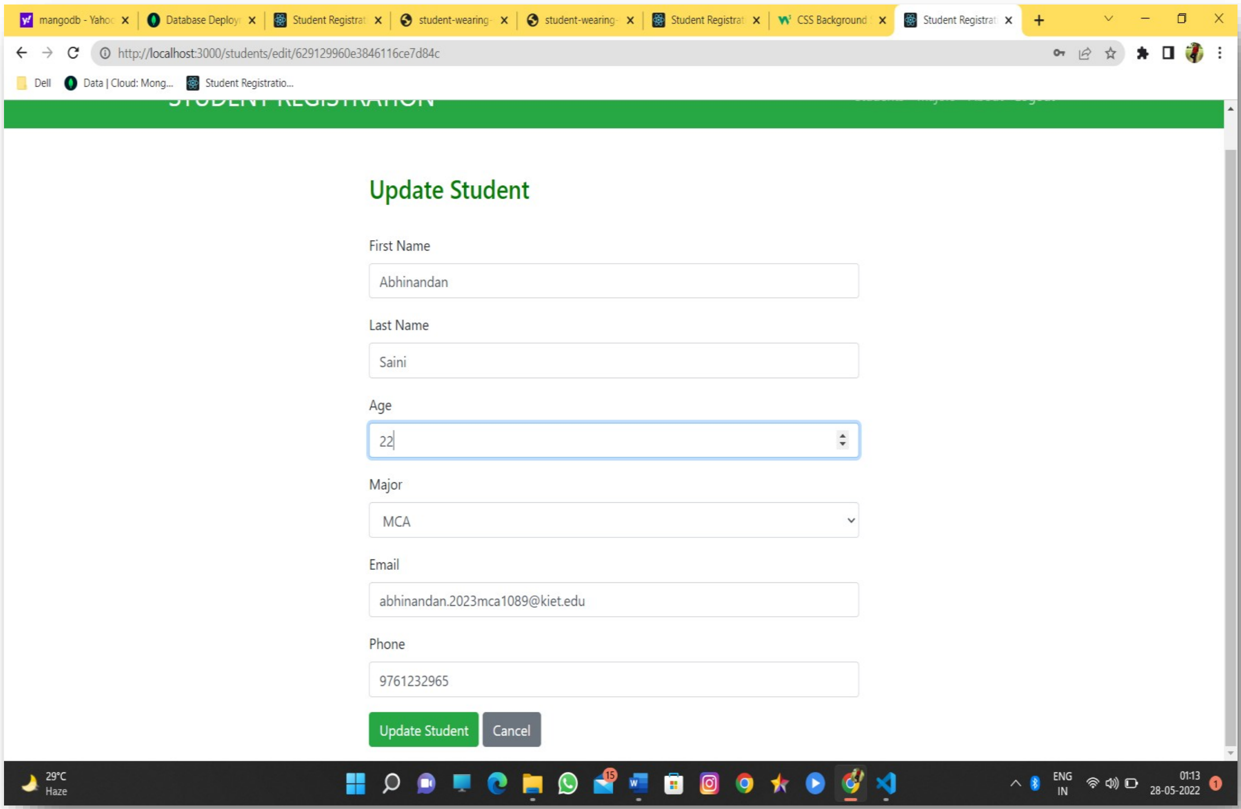
Phone: 9761232965



http://localhost:3000/students/edit/629129960e3846116ce7d84c

Update Student

First Name	Abhinandan
Last Name	Saini
Age	21
Major	MCA
Email	abhinandan.2023mca1089@kiet.edu
Phone	9761232965



Update Student

First Name

Abhinandan

Last Name

Saini

Age

22

Major

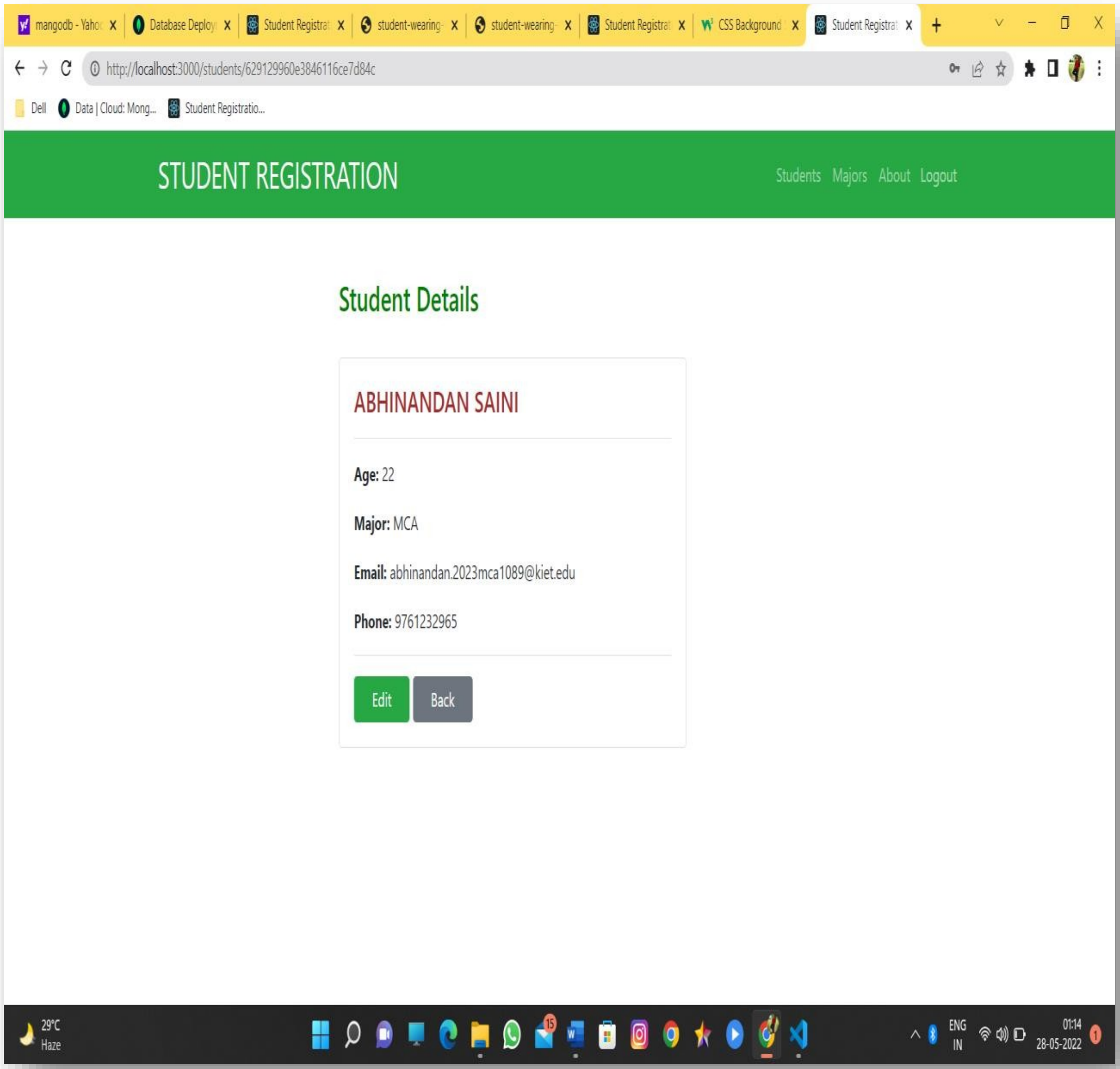
MCA

Email

abhinandan.2023mca1089@kiet.edu

Phone

9761232965



Student Details

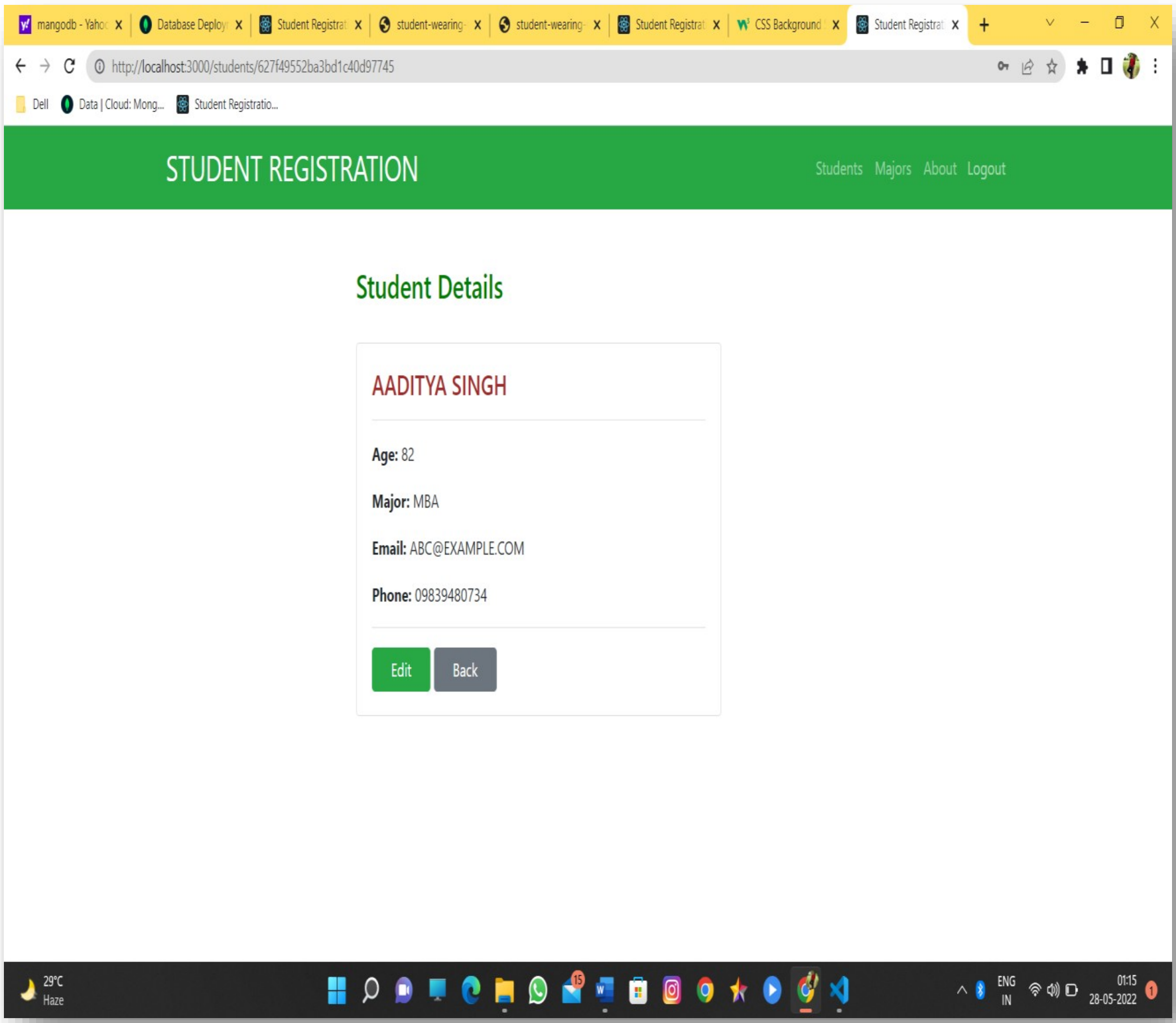
ABHINANDAN SAINI

Age: 22

Major: MCA

Email: abhinandan.2023mca1089@kiet.edu

Phone: 9761232965



STUDENT REGISTRATION

Student Details

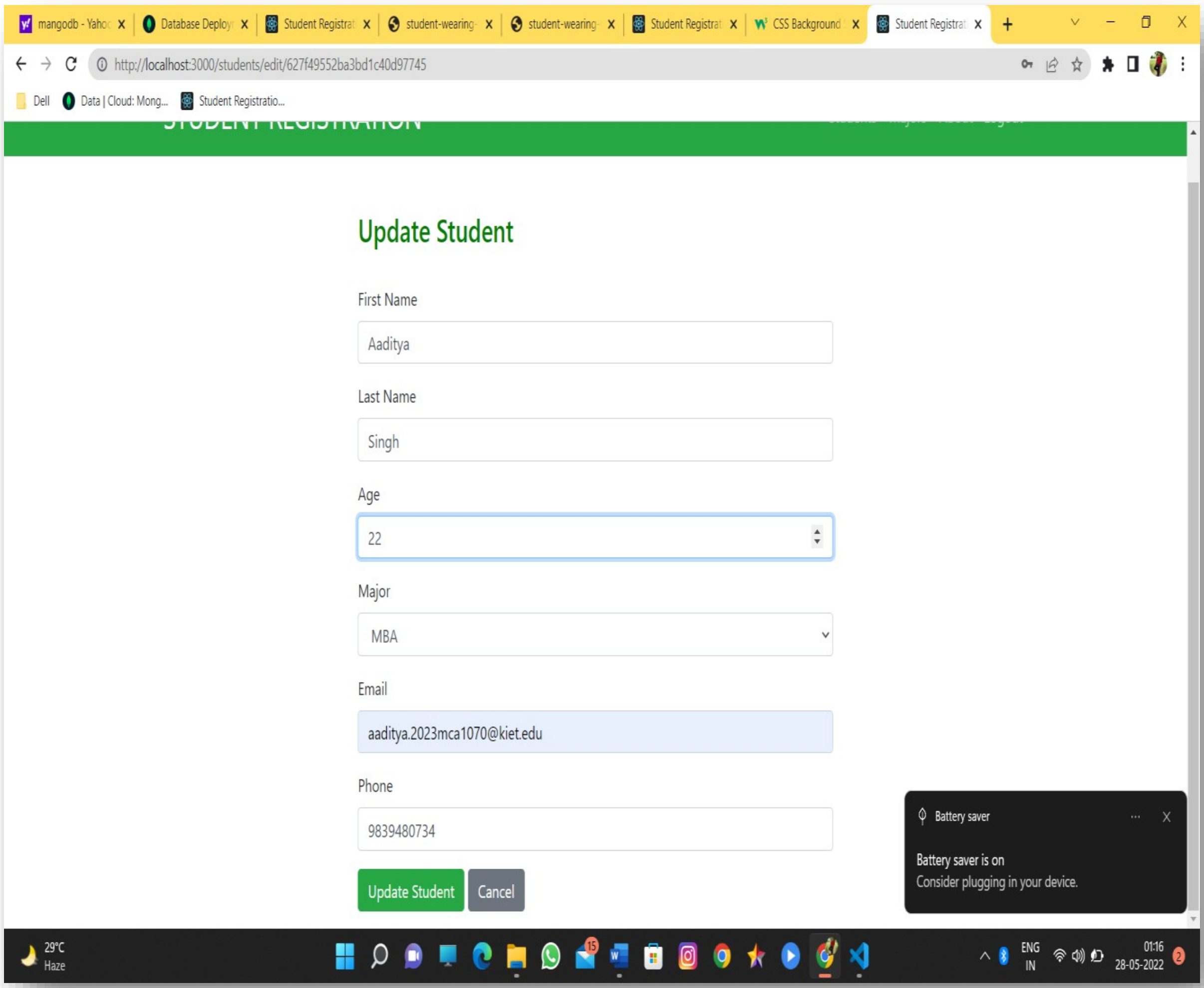
AADITYA SINGH

Age: 82

Major: MBA

Email: ABC@EXAMPLE.COM

Phone: 09839480734



http://localhost:3000/students/edit/627f49552ba3bd1c40d97745

Update Student

First Name

Aaditya

Last Name

Singh

Age

22

Major

MBA

Email

aaditya.2023mca1070@kiet.edu

Phone

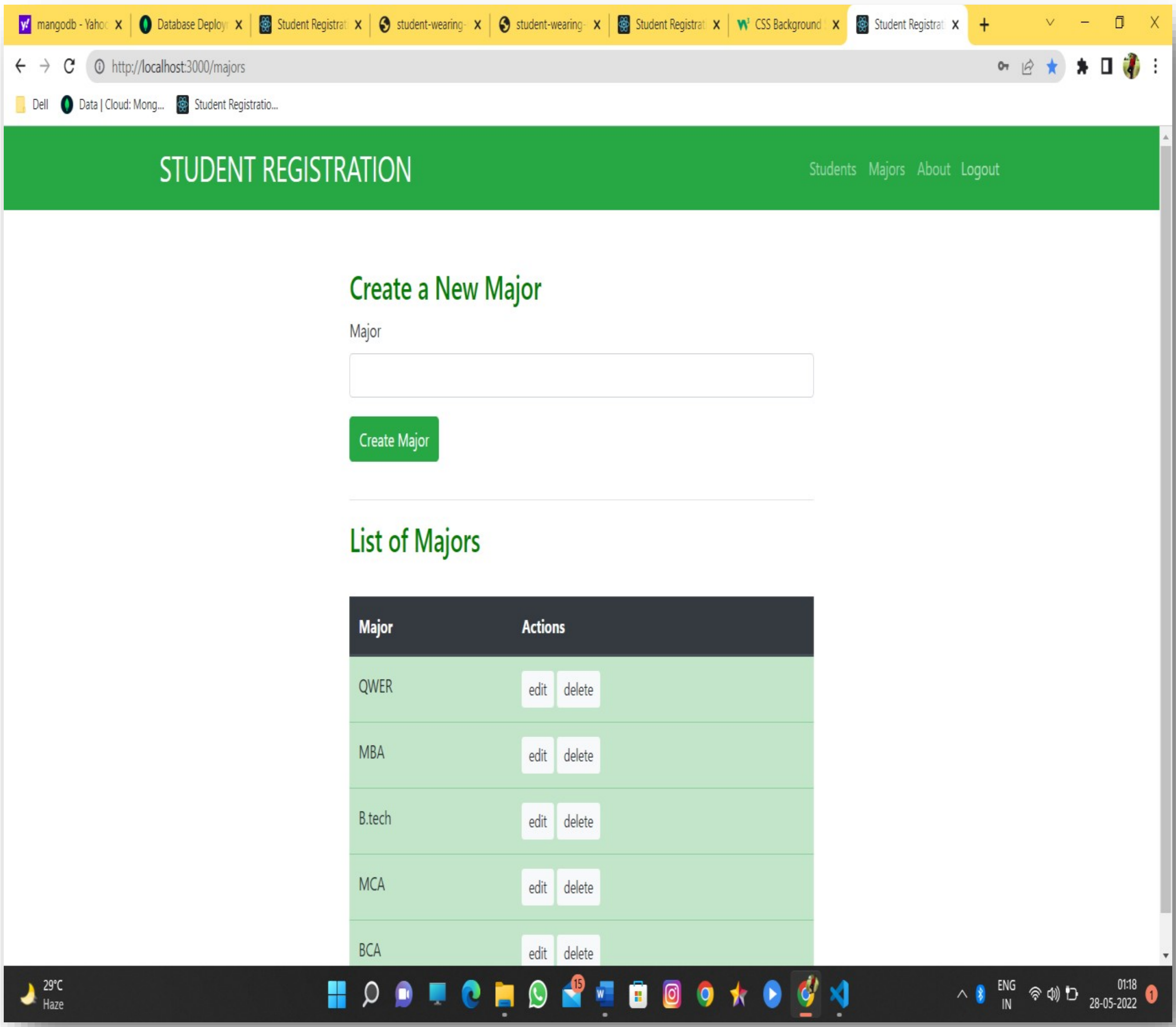
9839480734

Update Student Cancel

Battery saver

Battery saver is on

Consider plugging in your device.



STUDENT REGISTRATION

Students Majors About Logout

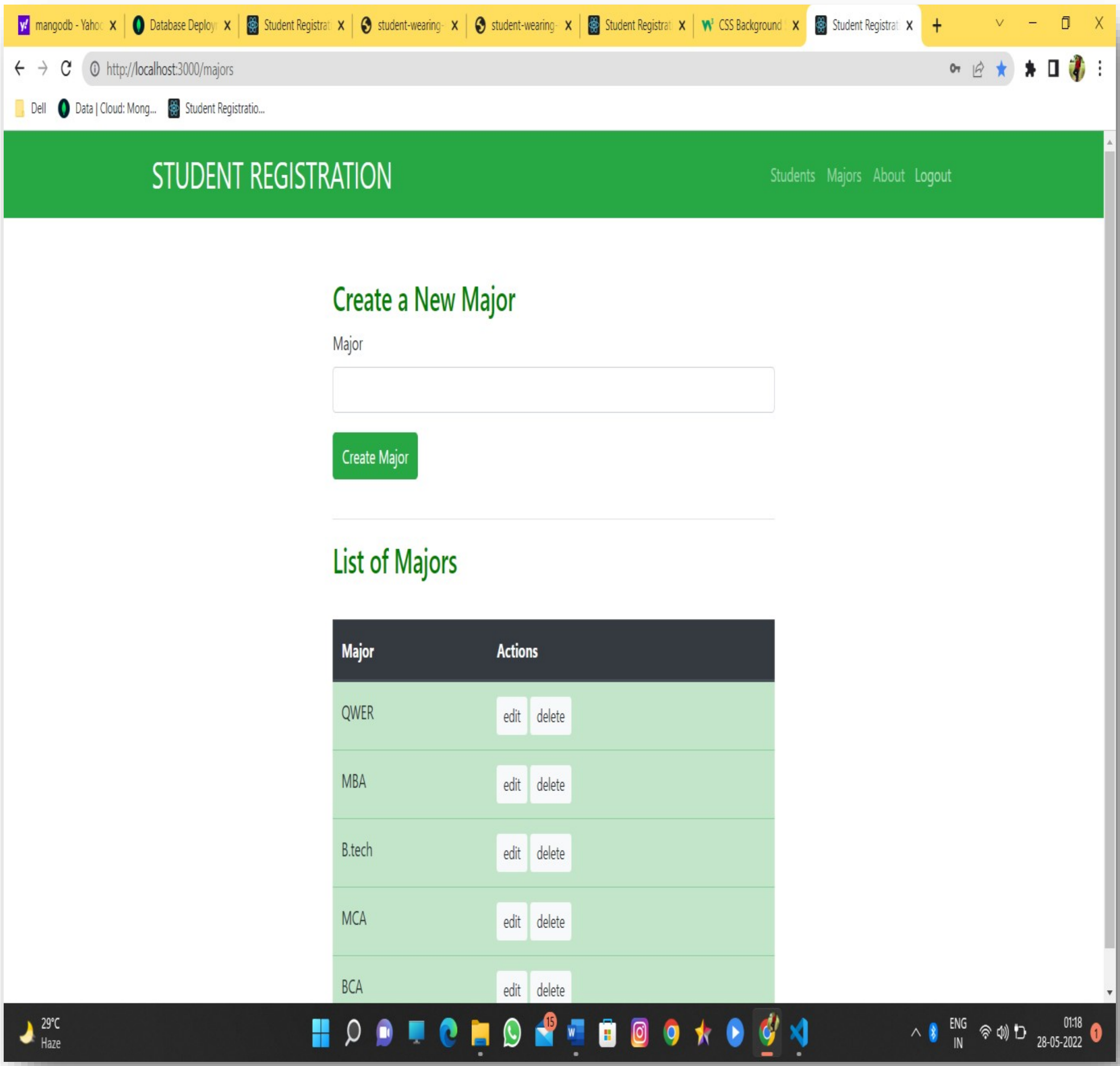
Create a New Major

Major

Create Major

List of Majors

Major	Actions
QWER	edit delete
MBA	edit delete
B.tech	edit delete
MCA	edit delete
BCA	edit delete



STUDENT REGISTRATION

Students Majors About Logout

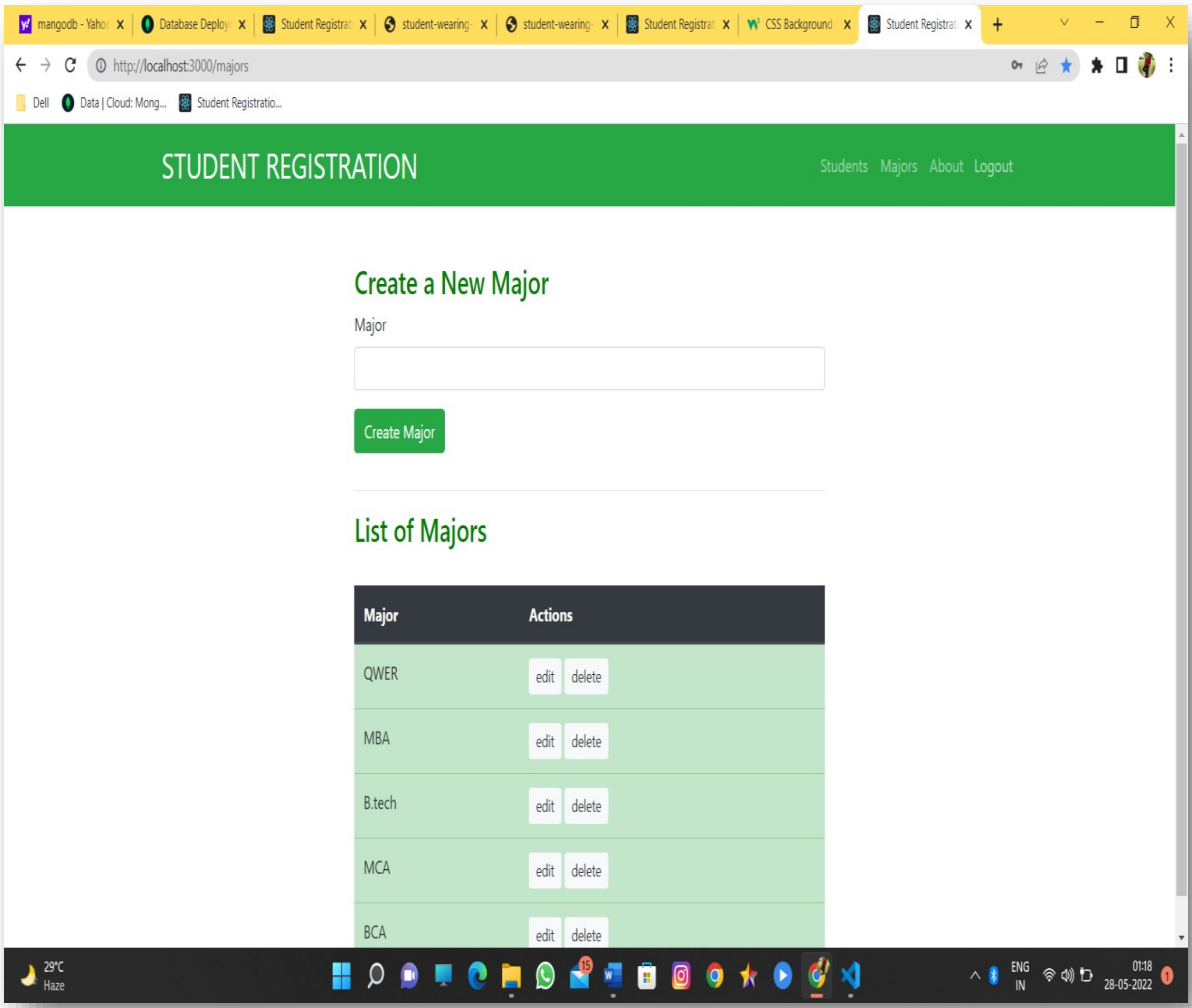
Create a New Major

Major

Create Major

List of Majors

Major	Actions
QWER	edit delete
MBA	edit delete
B.tech	edit delete
MCA	edit delete
BCA	edit delete



http://localhost:3000/majors

STUDENT REGISTRATION

Students Majors About Logout

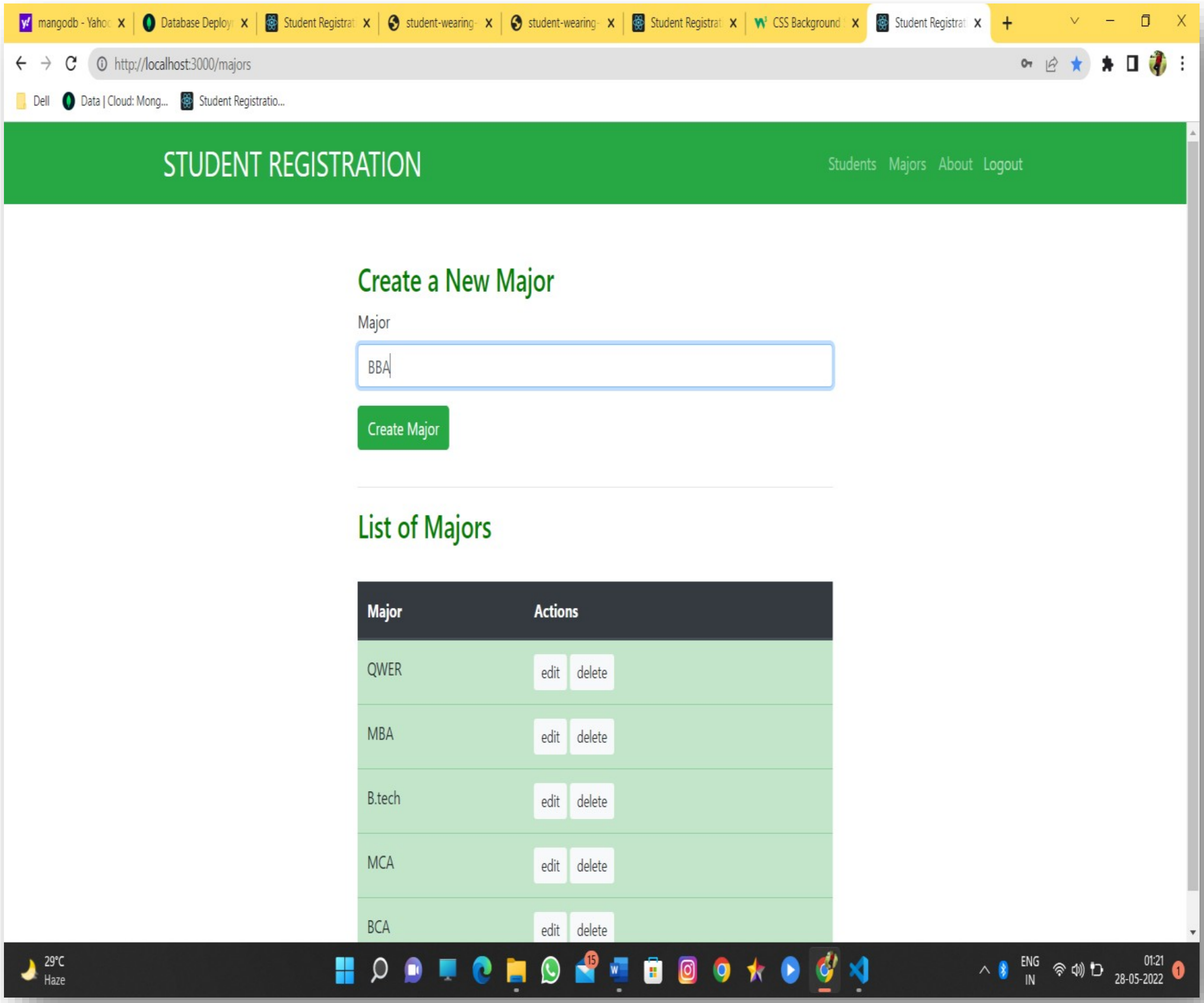
Create a New Major

Major

Create Major

List of Majors

Major	Actions
QWER	edit delete
MBA	edit delete
B.tech	edit delete
MCA	edit delete
BCA	edit delete



Create a New Major

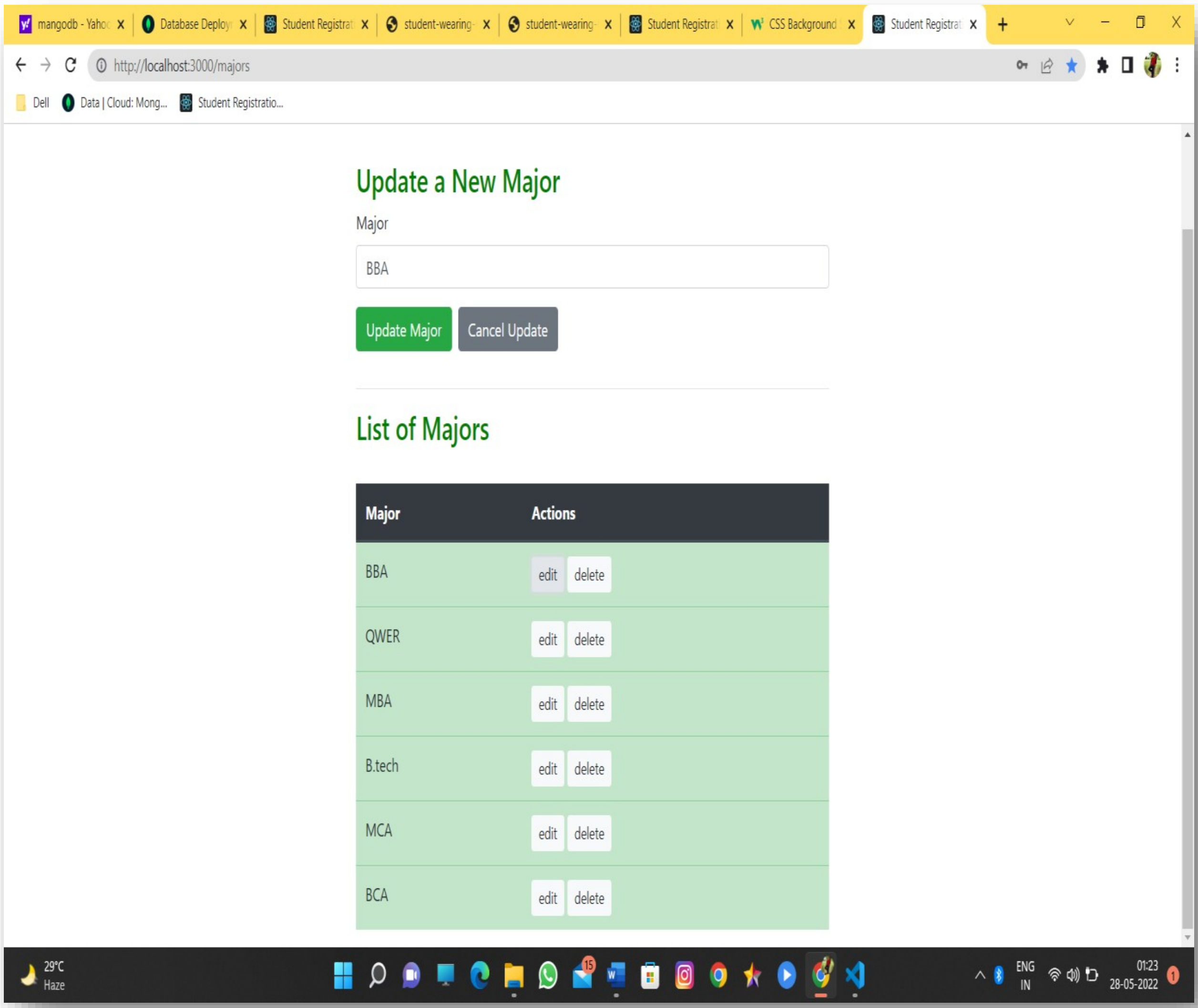
Major

BBA

Create Major

List of Majors

Major	Actions
QWER	edit delete
MBA	edit delete
B.tech	edit delete
MCA	edit delete
BCA	edit delete



Update a New Major

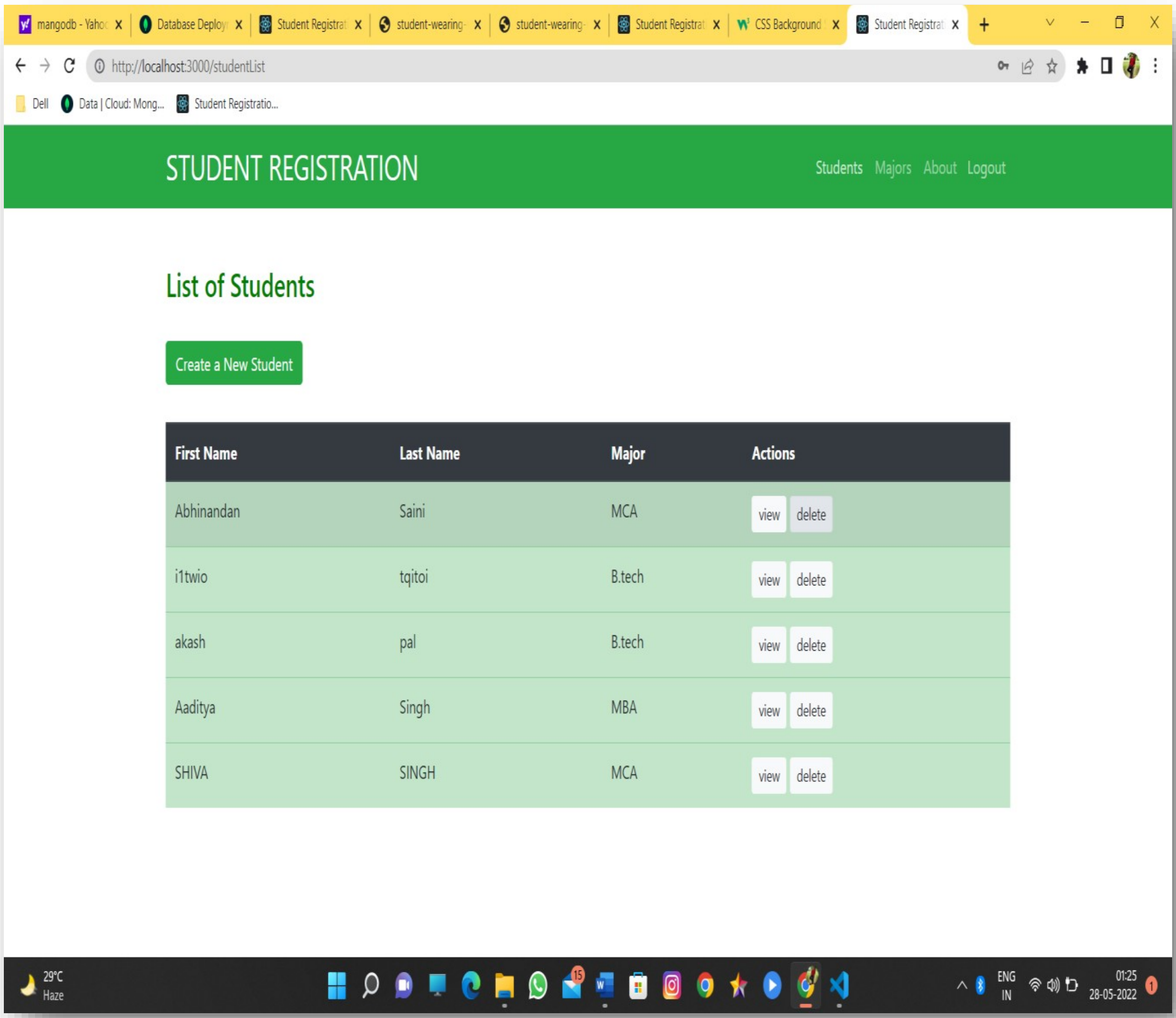
Major

BBA

Update Major | Cancel Update

List of Majors

Major	Actions
BBA	edit delete
QWER	edit delete
MBA	edit delete
B.tech	edit delete
MCA	edit delete
BCA	edit delete



STUDENT REGISTRATION

List of Students

First Name	Last Name	Major	Actions
Abhinandan	Saini	MCA	view delete
i1twio	tqitoi	B.tech	view delete
akash	pal	B.tech	view delete
Aaditya	Singh	MBA	view delete
SHIVA	SINGH	MCA	view delete

Testing

Test Cases

- **Black box Testing**: is the testing process in which tester can perform testing on an application without having any internal structural knowledge of application.

  Usually Test Engineers are involved in the black box testing.

- **White box Testing**: is the testing process in which tester can perform testing on an application with having internal structural knowledge.

  Usually The Developers are involved in white box testing.

- **Gray Box Testing**: is the process in which the combination of black box and white box techniques are used.

- **Smoke Testing**: is the process of initial testing in which tester looks for the availability of all the functionality of the application in order to perform detailed testing on them. (Main check is for available forms)
- **Sanity Testing**: is a type of testing that is conducted on an application initially to check for the proper behavior of an application that is to check all the functionality are available before the detailed testing is conducted by on them.
- **Regression Testing**: is one of the best and important testing. Regression testing is the process in which the functionality, which is already tested before, is once again tested whenever some new change is added in order to check whether the existing functionality remains same.

- **Re-Testing**: is the process in which testing is performed on some functionality which is already tested before to make sure that the defects are reproducible and to rule out the environments issues if at all any defects are there.
- **Static Testing**: is the testing, which is performed on an application when it is not been executed. ex: GUI, Document Testing

- **Alpha Testing**: it is a type of user acceptance testing, which is conducted on an application when it is just before released to the customer.
- **Beta-Testing**: it is a type of UAT that is conducted on an application when it is released to the customer, when deployed in to the real time environment and being accessed by the real time users.
- **Monkey Testing**: is the process in which abnormal operations, beyond capacity operations are done on the application to check the stability of it in spite of the users abnormal behavior.
- **Compatibility testing**: it is the testing process in which usually the products are tested on the environments with different combinations of databases (application servers, browsers…etc) In order to check how far the product is compatible with all these environments platform combination.
- **Installation Testing**: it is the process of testing in which the tester try to install or try to deploy the module into the corresponding environment by following the guidelines produced in the deployment document and check whether the installation is successful or not.
- **Adhoc Testing**: Adhoc Testing is the process of testing in which unlike the formal testing where in test case document is used, with out that test case document testing can be done of an application, to cover that testing of the future which are not covered in that test case document. Also it is intended to perform GUI testing which may involve the cosmatic issues.

TCD (Test Case Document):

Test Case Document Contains

- Test Scope (or) Test objective
- Test Scenario
- Test Procedure
- Test case

This is the sample test case document for the Acadamic details of student project:

Test scope:

- Test coverage is provided for the screen “ Acadamic status entry” form of a student module of university management system application
- Areas of the application to be tested

Test Scenario:

- When the office personals use this screen for the marks entry, calculate the status details, saving the information on student’s basis and quit the form.

Test Procedure:

- The procedure for testing this screen is planned in such a way that the data entry, status calculation functionality, saving and quitting operations are tested in terms of Gui testing, Positive testing, Negative testing using the corresponding Gui test cases, Positive test cases, Negative test cases respectively