SCHOOL MANAGEMENT SYSTEM

GROUP PROJECT

GROUP : 5

GROUP MEMBERS

ARITRA DUTTA
2054088

SOHINI GHOSH
2054089

ARITRICK BHOWMICK
2054075

DEPARTMENT OF INFORMATION TECHNOLOGY HERITAGE INSTITUTE OF TECHNOLOGY

(private autonomous college affiliated to MAKAUT, West Bengal)

BONAFIDE CERTIFICATE

This to certify that the project report entitled 'School Management System' is the bonafide record of project work done by Aritra Dutta, Sohini Ghosh and Aritrick Bhowmick of the department of Information Technology in the year 2022 for the partial fulfilment of the requirements of the

Software Engineering Laboratory

Faculty Signature

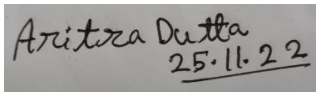
H.O.D

ACKNOWLDGEMENT

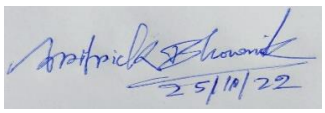
There is no scope of learning and improvement till one makes a mistake. We take this opportunity to express our gratitude and deep regards to teachers Asst. Prof Subhajit Rakshit, Asst. Prof Shantanu Ghosh, Asst. Prof Siddharth Bose for guiding, mentoring, encouragement and constantly helping us with innovative and constructive ideas. This project is nothing but a return to all the teaching that they have provided us with.

We are obliged to all our teachers for the valuable information provided by them in their respective fields and we are grateful for their co-operation during the period of assignment.

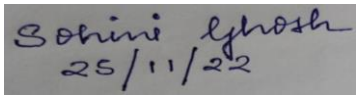
Thank you. We shall keep working on each and every flaws and present the best in future.



Aritra Dutta



Aritrick Bhowmick



Sohini Ghosh

INDEX

Hardware and Software Requirements

Hardware requirements :

- Intel Pentium or above
- Windows 7 or above
- Proper server to host the website

Software requirements :

- Supported browser
  - Chrome
  - Fire-fox
  - Brave
  - Microsoft edge

SYSTEM
REQUIREMENTS SPECIFICATION
FOR

SCHOOL MANAGEMENT SYSTEM

prepared by

GROUP 5

MEMBER NAMES:

ARITRA DUTTA (2054088)

SOHINI GHOSH (2054089)

ARITRICK BHOWMICK (2054075)

Introduction

Purpose:

This project is targeted mainly for school managements, where they can easily handle their student's details and other day-to-day activities.

Project Scope:

Students can explore their school details as they can view or request updation of their own details, can view their attendance, can also pay fees as well as the administration can handle student details as well as their own and other requirements. This project is restricted only to school managements and their purposes.

The fee payment part has a restriction while paying fees as the project has no bank account linked to and thus real time transactions are not possible. However, when linked with a bank it will be possible.

References:

Color Lib : https://colorlib.com/

Font Awesome : https://fontawesome.com/

W3C Templates : https://www.w3.org/

Free CSS : https://www.free-css.com/

System Description

Students can explore their school details as they can view or request updation of their own details, can view their attendance, can also pay fees as well as the administration can handle student details as well as their own and other

requirements. Student or Admin can change their password if required. Admin can update their own details as well as student's details'. A student as well as the Admin both can view the fee details of a student and a student can generate the fee receipt. It was also asked if admin can update the attendance.

System Features

System Feature 1

Login for Student

Login for Admin

System Feature 2

Forgot/Change password for student

Forgot/Change password for admin

System Feature 3

Request edit profile for

Student Edit profile for Admin

System Feature 4

Pay Fees/View Fee Details for Student

View Fee Details of Student by Admin

System Feature 5

View Attendance by Student

Listing of the student/Add Student by Admin

Database Dictionary

student1 Table

Attribute Name	Type	Null?
Roll (Primary KEY)	int(100)	NOT NULL
Class (Primary Key)	bigint(100)	NOT NULL
Email(Foreign Key)	varchar(100)	NOT NULL
Name	varchar(100)	NOT NULL
Phone_Number	varchar(100)	NOT NULL
Address	varchar(100)	NOT NULL
Blood_Group	varchar(100)	NOT NULL
Gender	varchar(100)	NOT NULL
DOB	date	NOT NULL
Payment	varchar(100)	NOT NULL

student_info Table

Attribute Name	Type	Null?
Roll	int(100)	NOT NULL
Class	bigint(100)	NOT NULL
Email(Primary Key)	varchar(100)	NOT NULL
Name	varchar(100)	NOT NULL
Password	varchar(100)	NOT NULL
Securecode	varchar(100)	NOT NULL

admin Table

Attribute Name	Type	Null?
Admin_id(Primary Key)	varchar(100)	NOT NULL
Name	varchar(100)	NOT NULL
Email	varchar(100)	NOT NULL
Password	varchar(100)	NOT NULL
Securecode	varchar(100)	NOT NULL
Phone_Number	varchar(10)	NOT NULL
Gender	varchar(100)	NOT NULL
Address	varchar(100)	NOT NULL

attendance Table

Attribute Name	Type	Null?
Roll (Primary KEY)	int(100)	NOT NULL
Class (Primary Key)	bigint(100)	NOT NULL
Date(Primary Key)	date	NOT NULL
Name	varchar(100)	NOT NULL
Email(Foreign Key)	varchar(100)	NOT NULL
Status	varchar(100)	NOT NULL

feedetails Table

Attribute Name	Type	Null?
Fees(Primary Key)	varchar(100)	NOT NULL
Class(Foreign Key)	varchar(100)	NOT NULL

paymentdetails Table

Attribute Name	Type	Null?
Transaction_Id	varchar(100)	NOT NULL
Roll	int(100)	NOT NULL
Class	bigint(100)	NOT NULL
Fees	varchar(100)	NOT NULL
Pay_Date	date	NOT NULL

External Interface Requirements

XAMPP(PHPMYADMIN)

VISUAL STUDIO CODE

MYSQL Database

Technical Requirements (Non functional)

Performance

Fetching a request takes 0.004-0.008 seconds.

Security

Encryption of data is required.

Usability

All type of users can easily use this software. The software is very student friendly thus enabling them to use it efficiently.

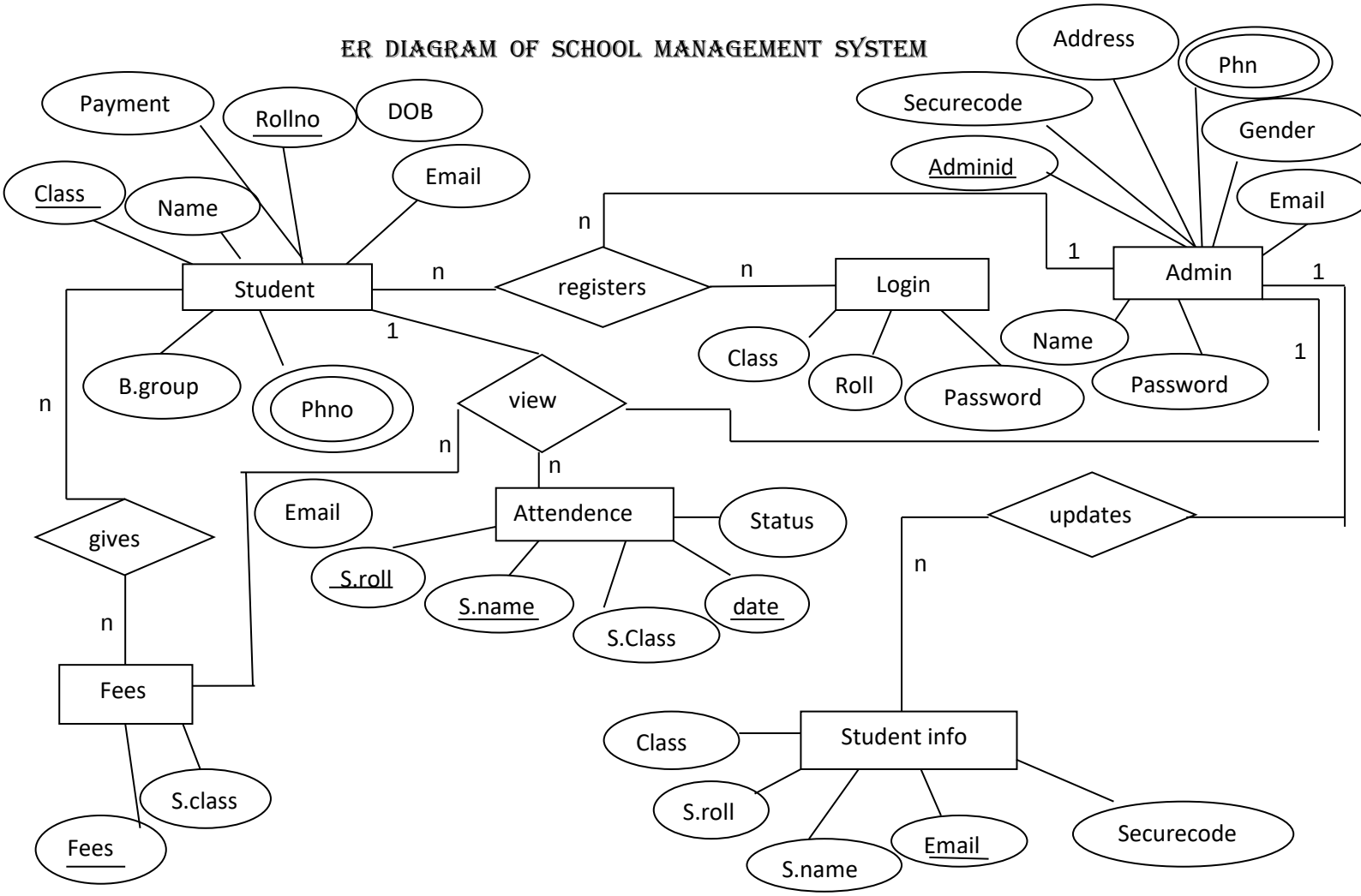
Maintainability

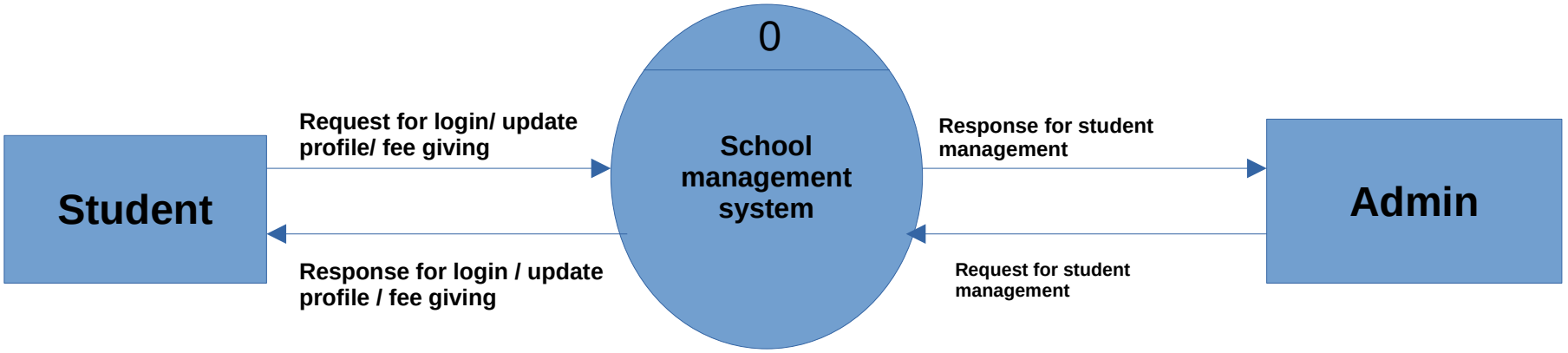
Easy to maintain and easily updatable.

Open Issues

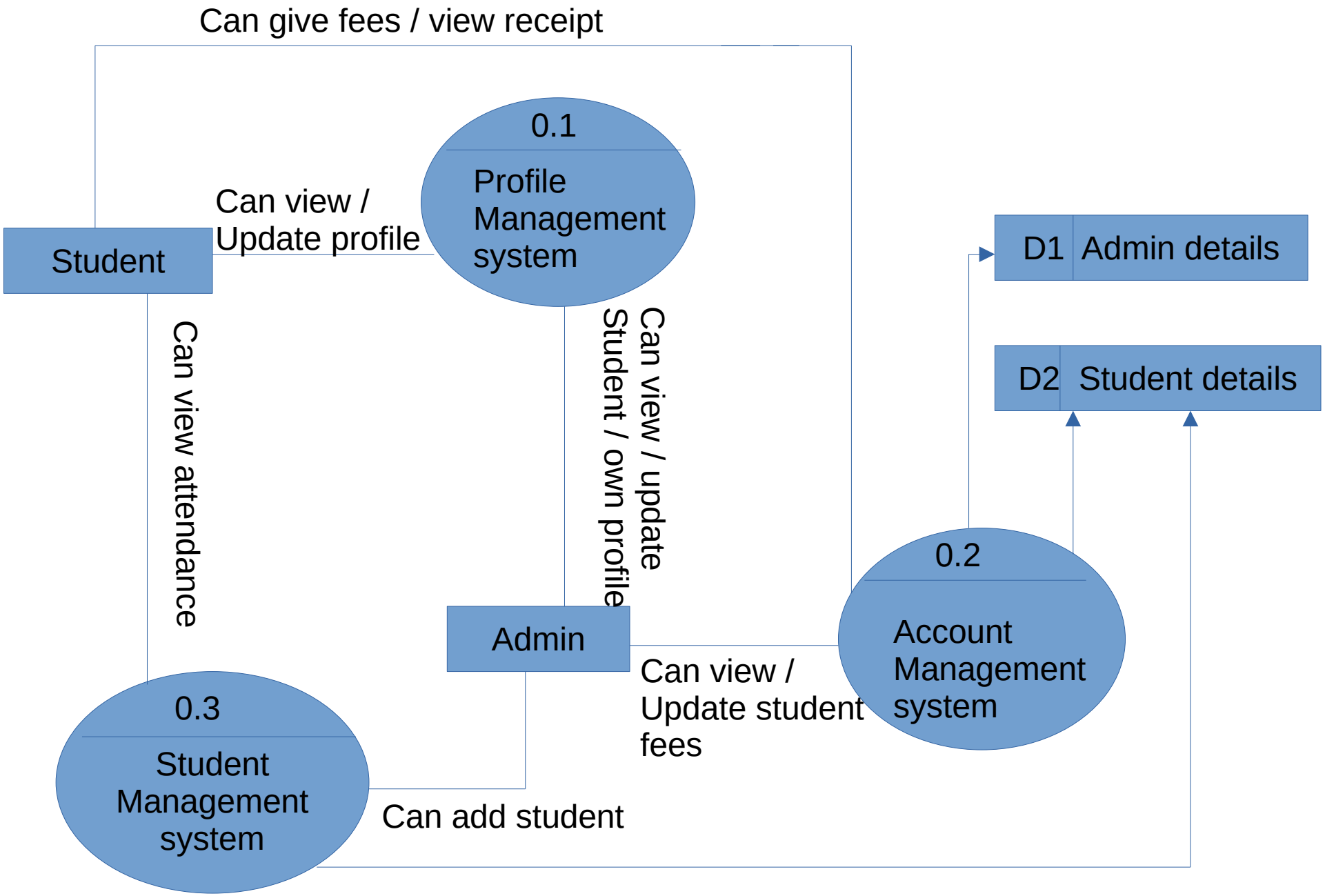
Frontend could be more attractive and there could be an option of Faculty management and Student Management System can be made more efficient

ER DIAGRAM OF SCHOOL MANAGEMENT SYSTEM

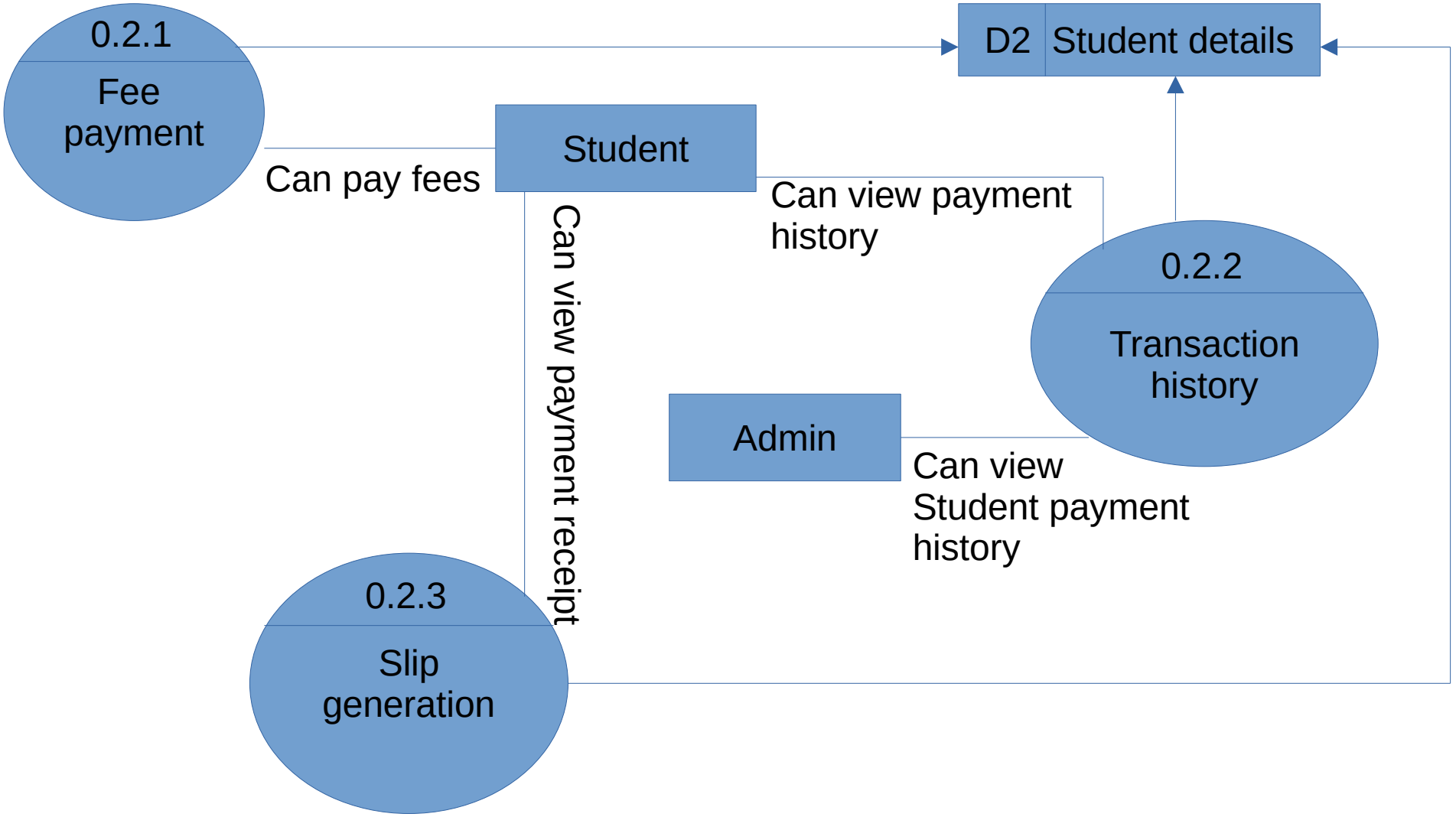



0
Student
Request for login/ update profile/ fee giving
Response for login / update profile / fee giving
School management system
Response for student management
Request for student management
Admin

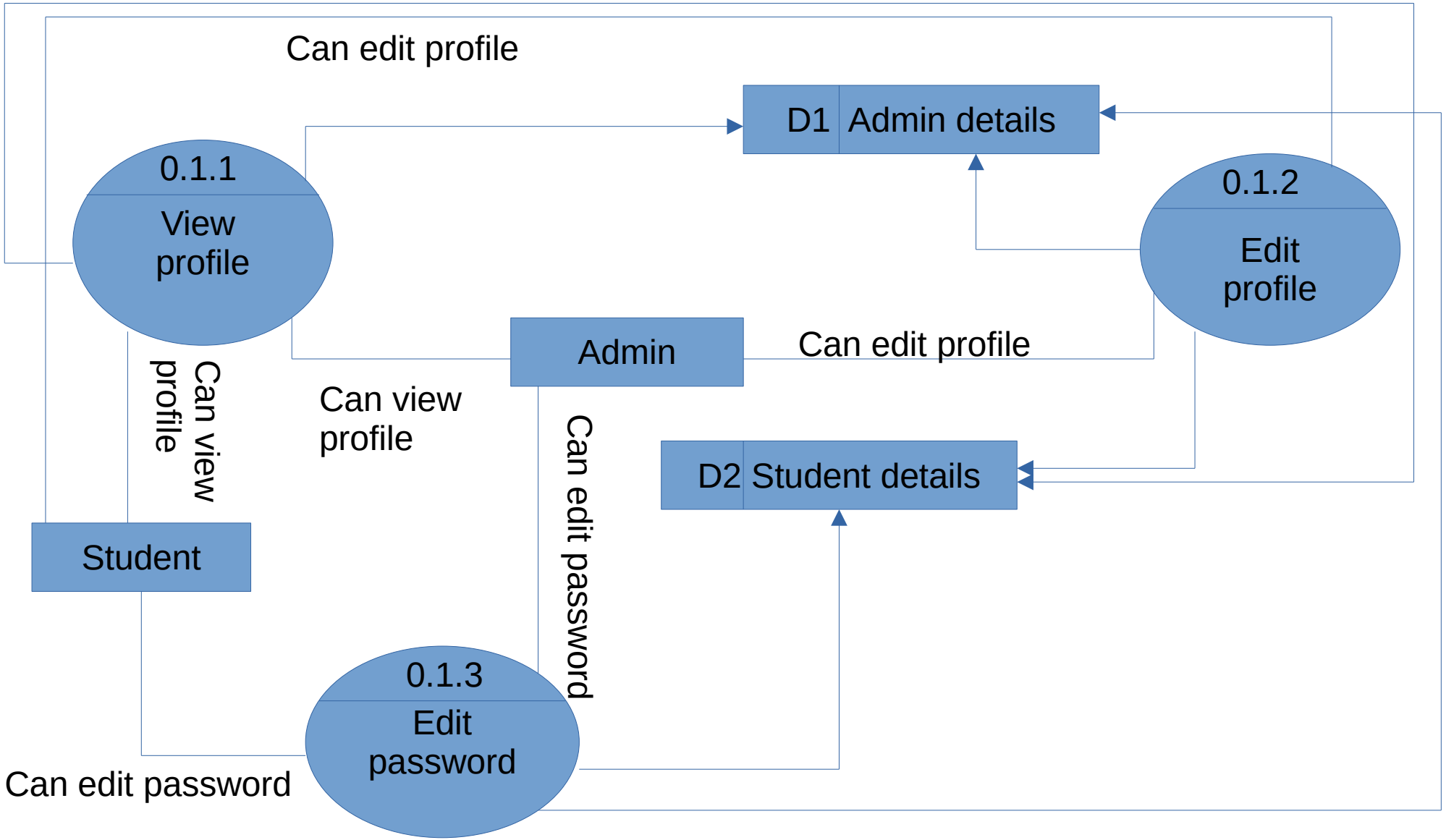
Level 0 DFD



1 level DFD of school management system

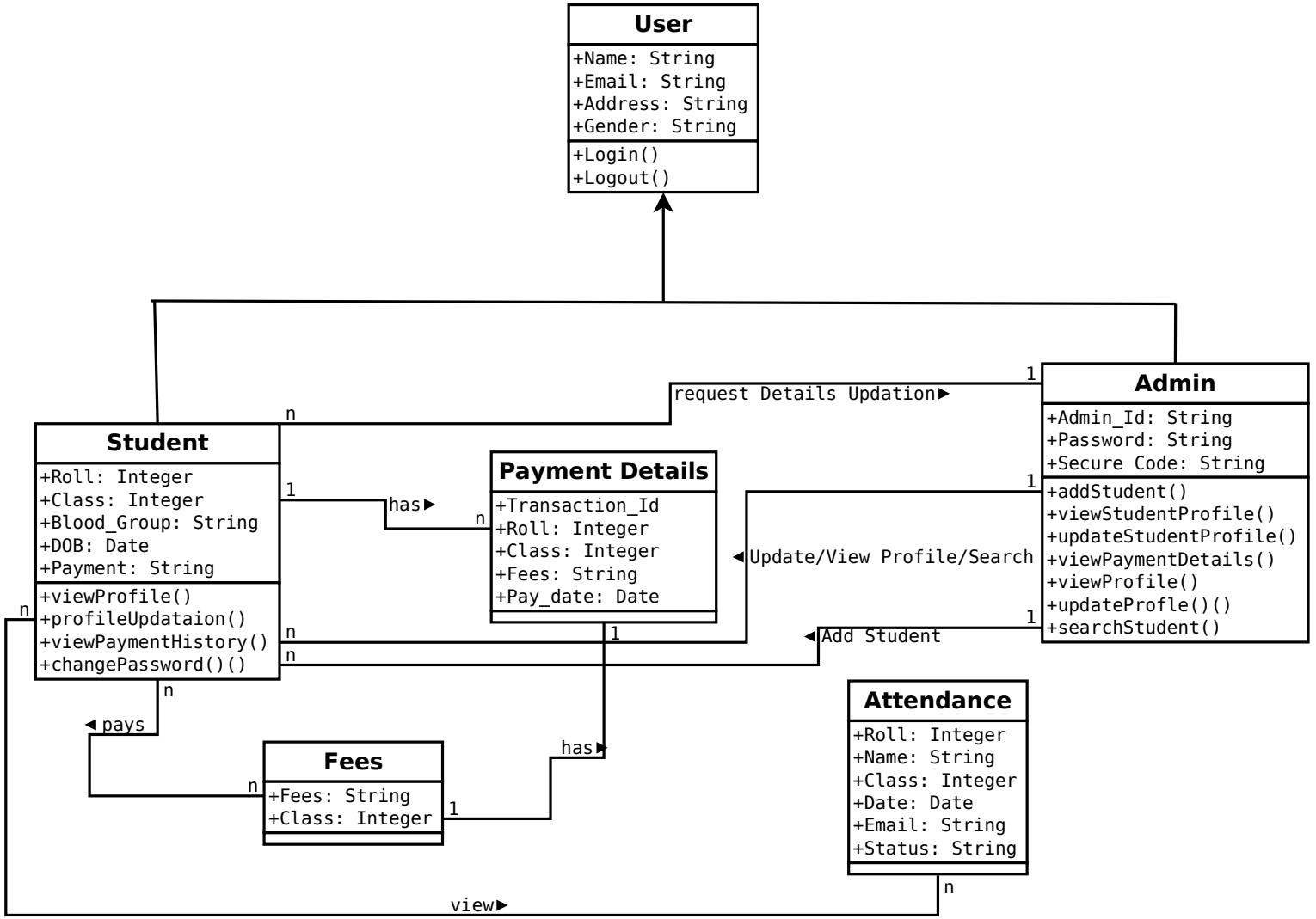


2 level DFD of school management system

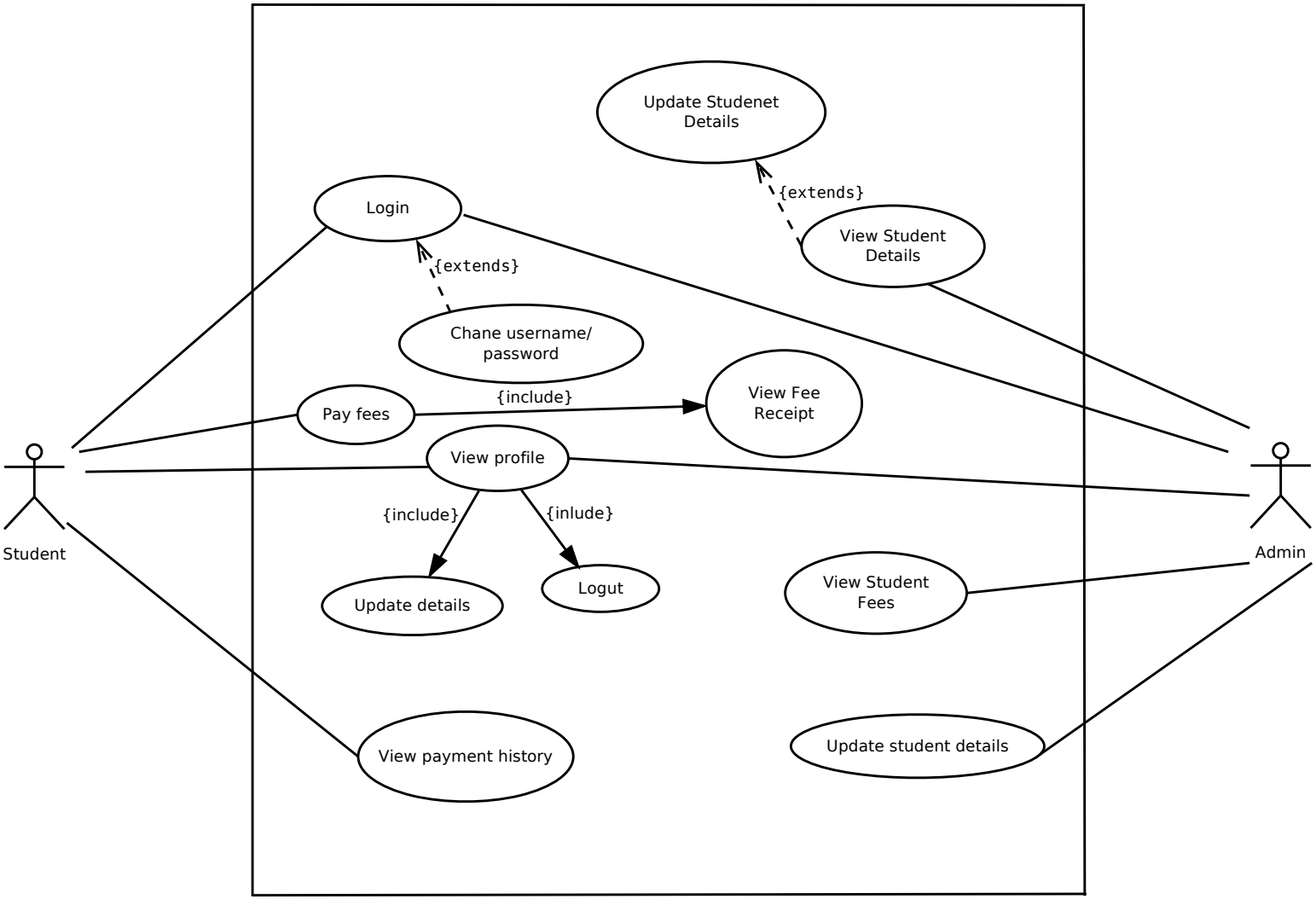


2 level DFD of school management system

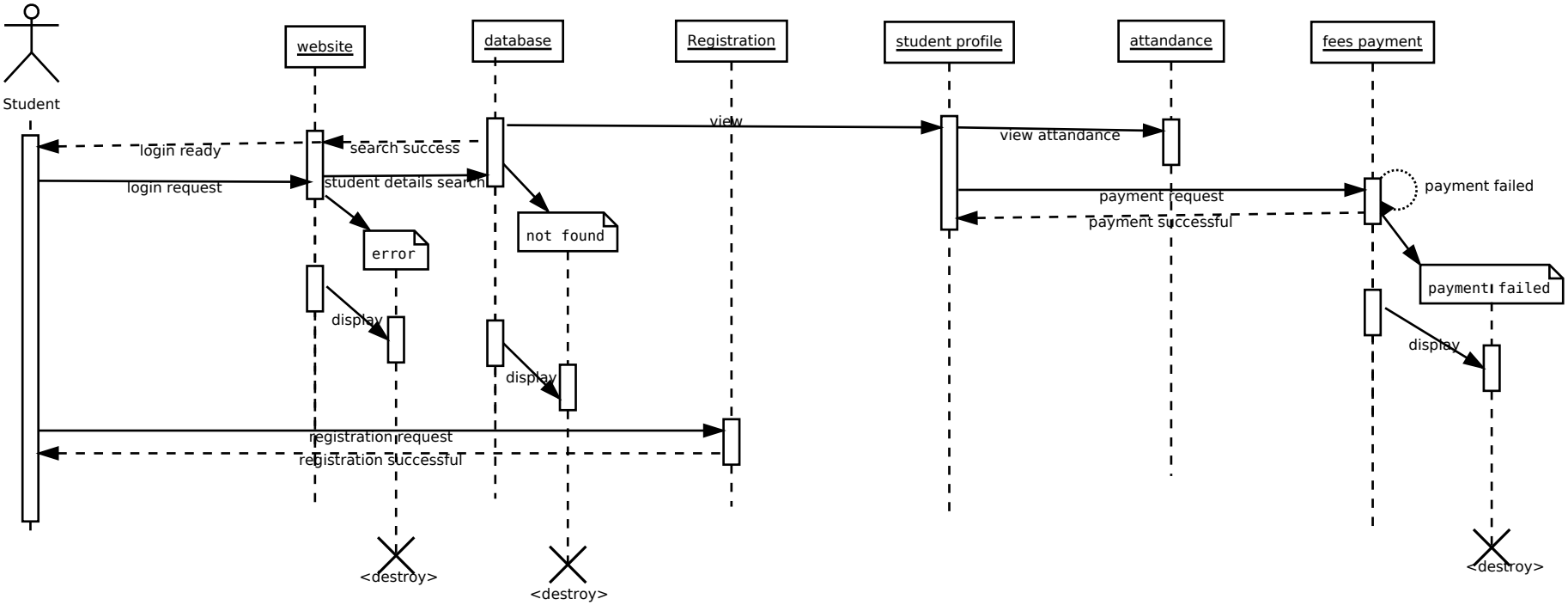
CLASS DIAGRAM



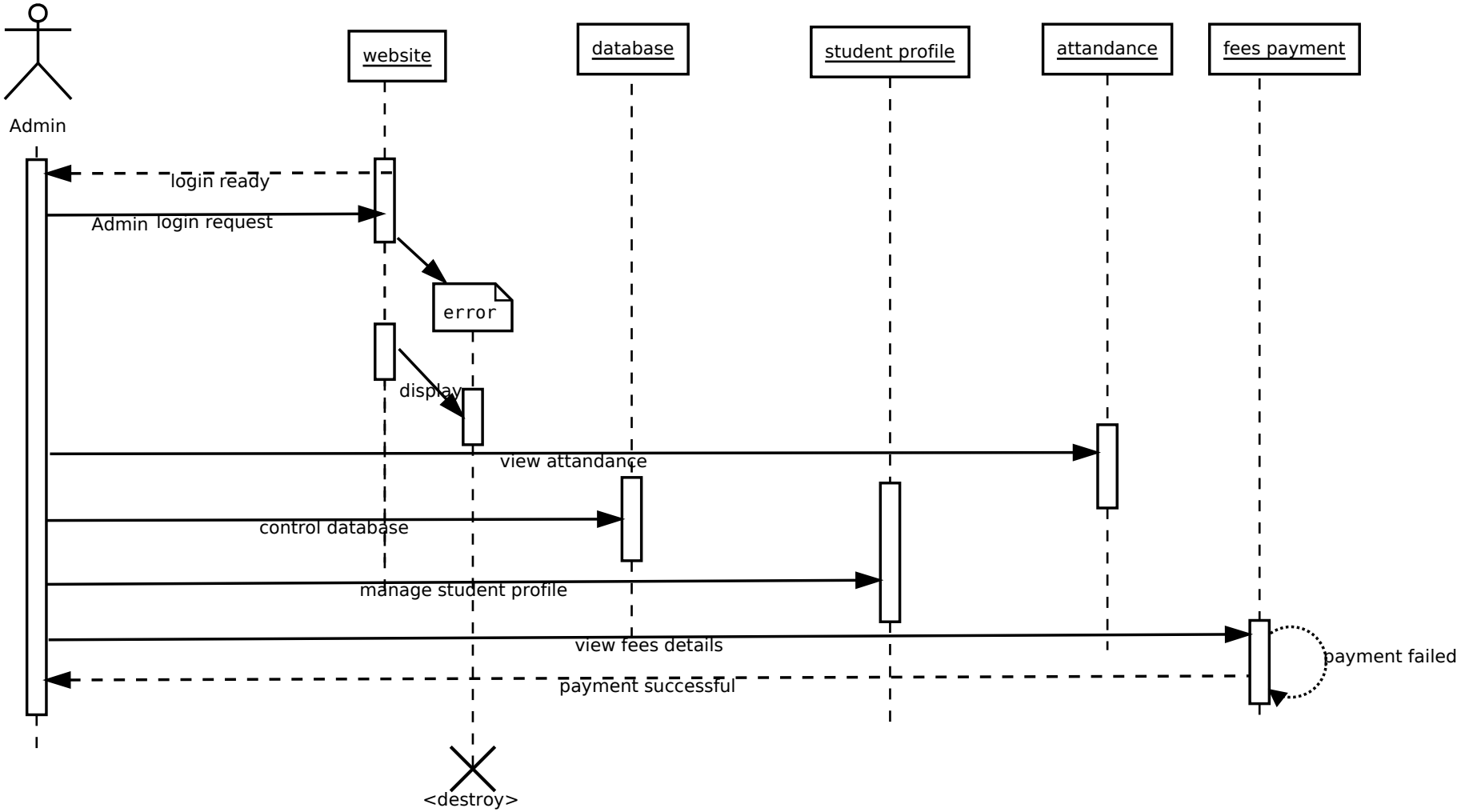
USE CASE DIAGRAM



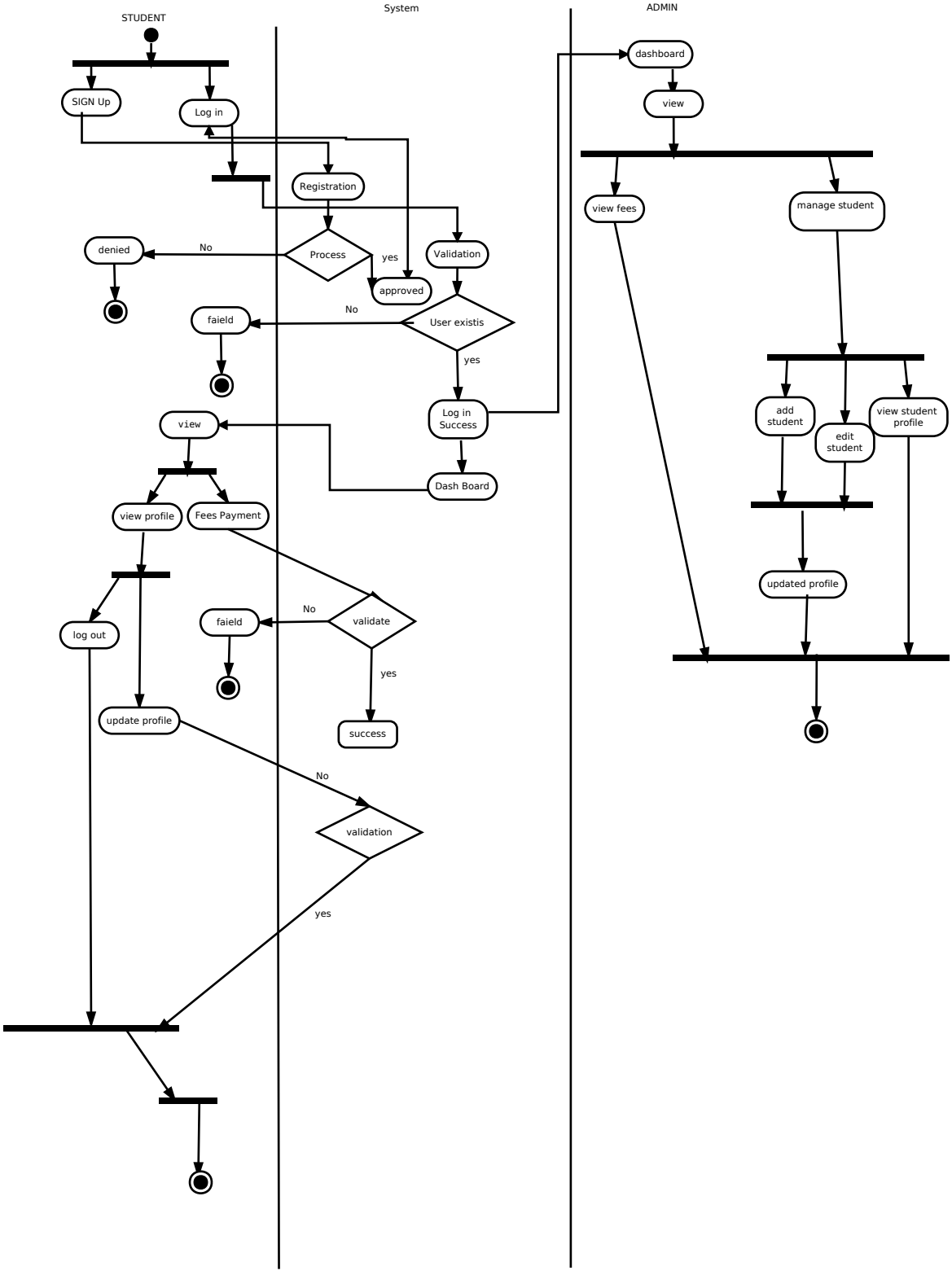
SEQUENCE DIAGRAM FOR STUDENT



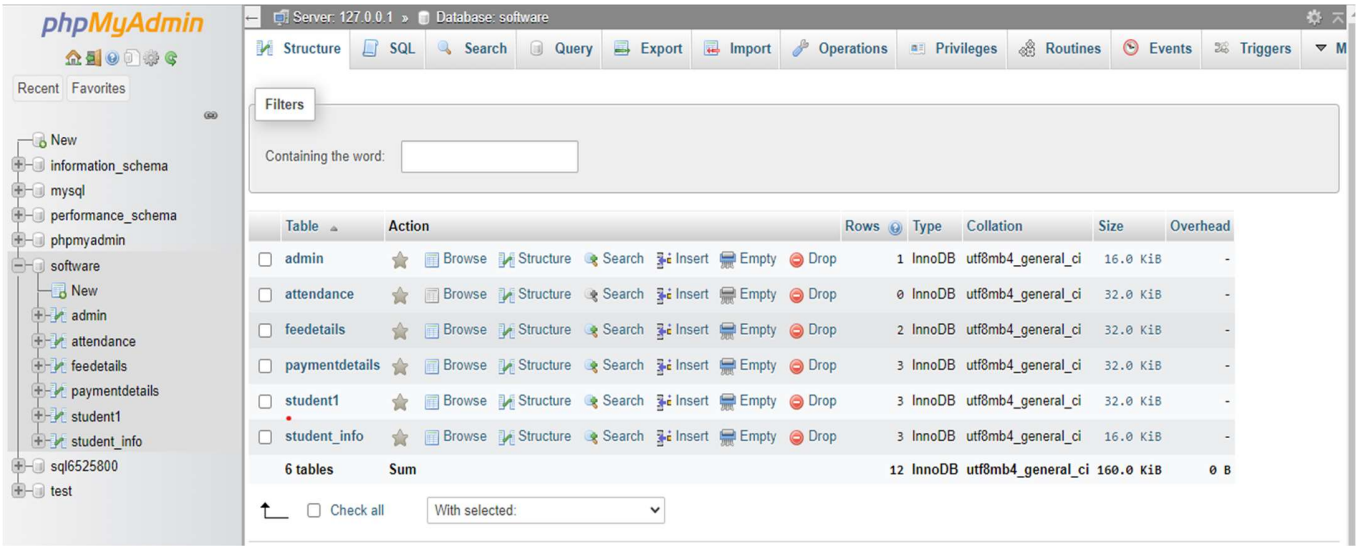
SEQUENCE DIAGRAM FOR ADMIN



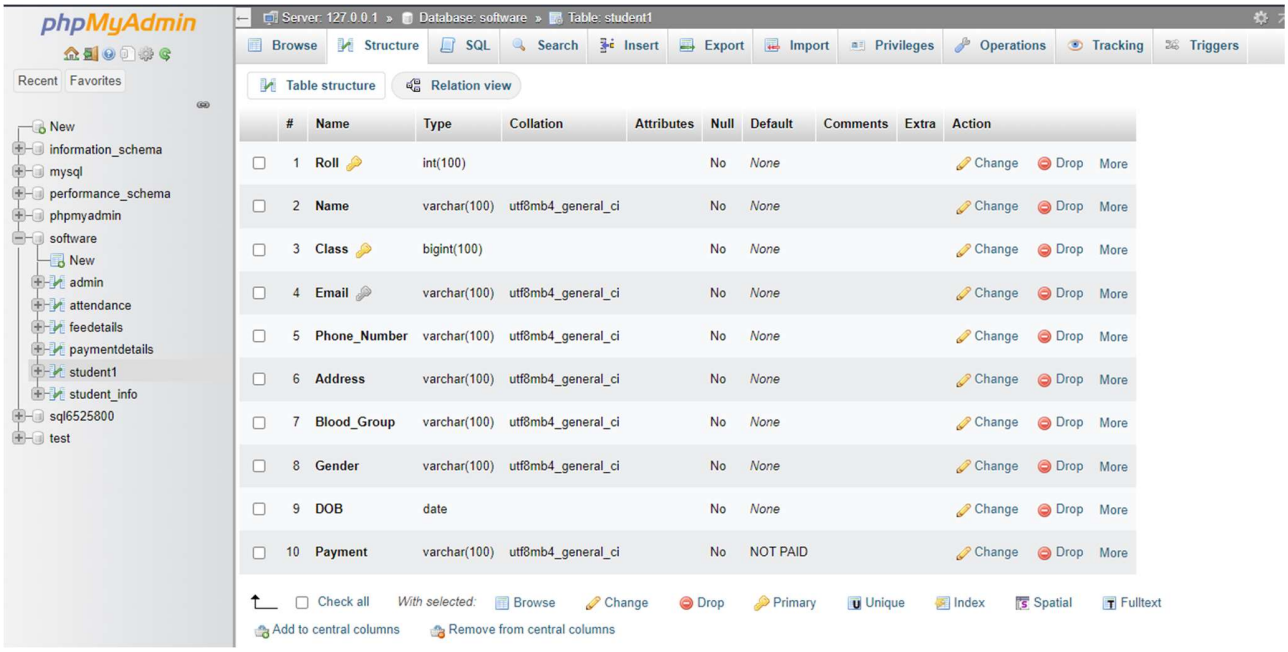
ACTIVITY DIAGRAM



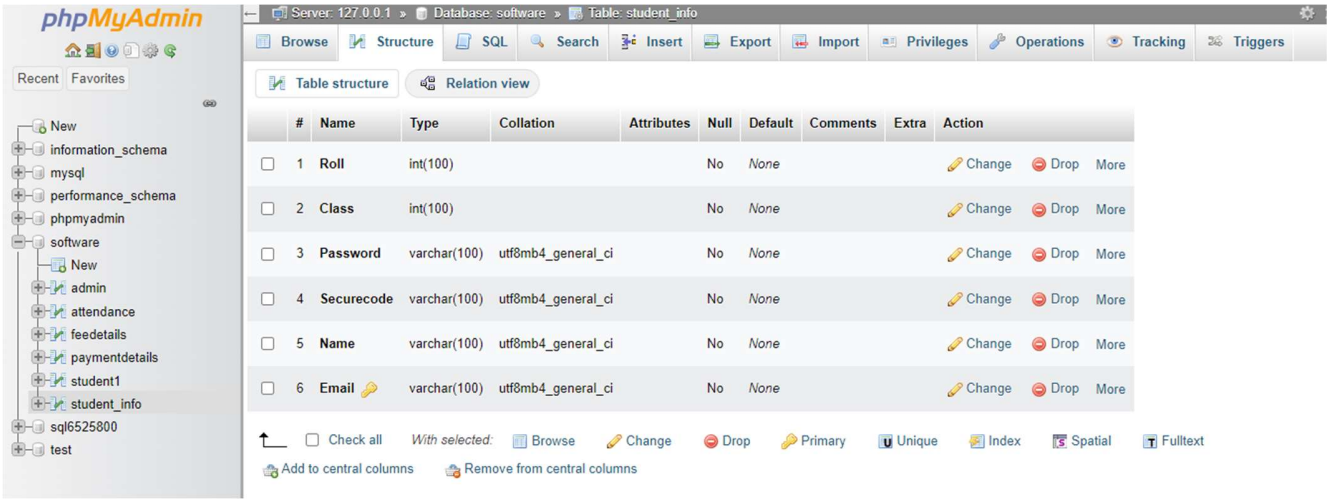
Database Design



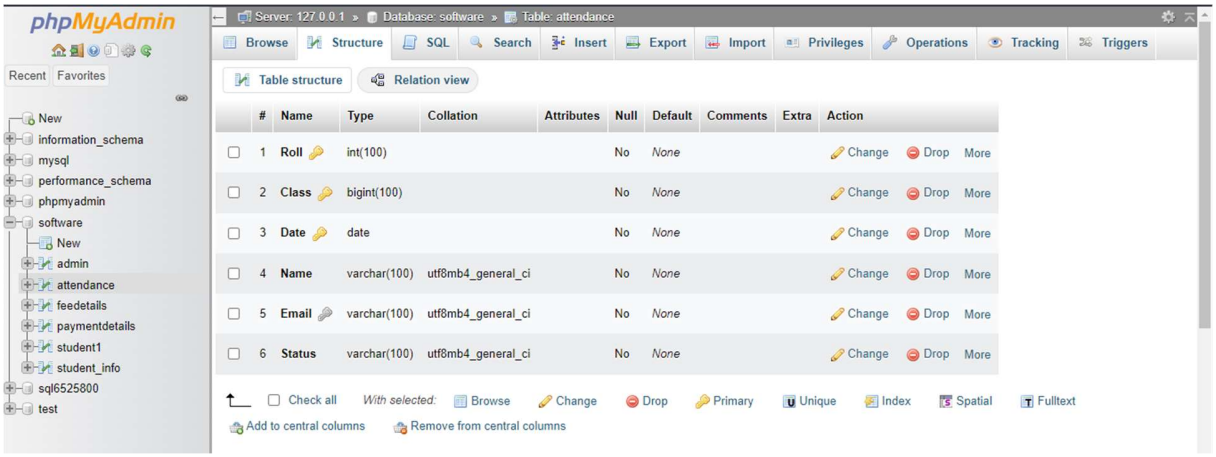
Data base tables



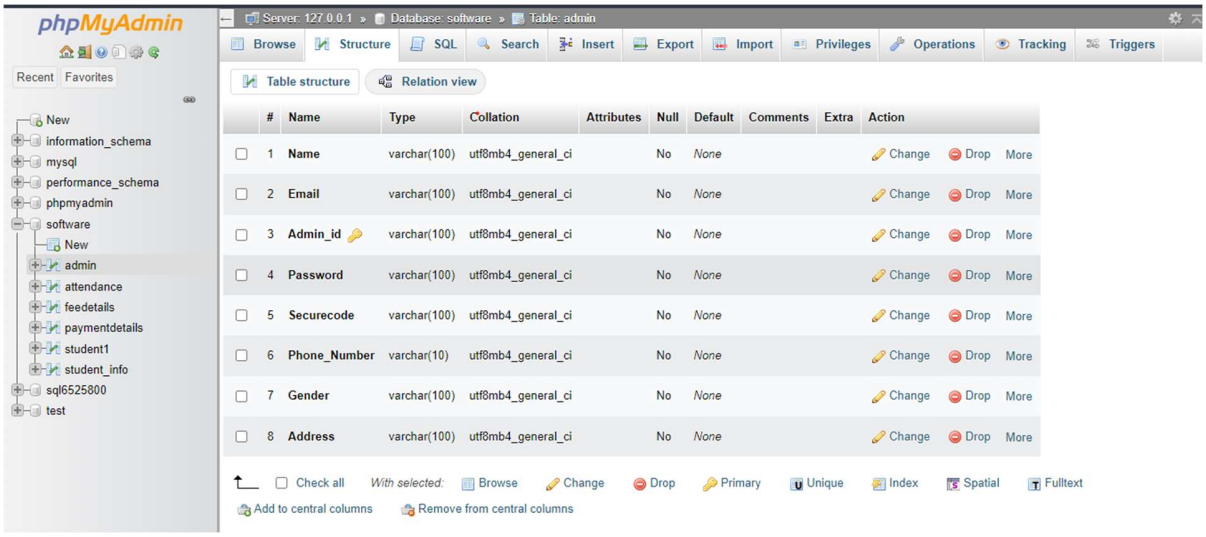
Structure of student1 table



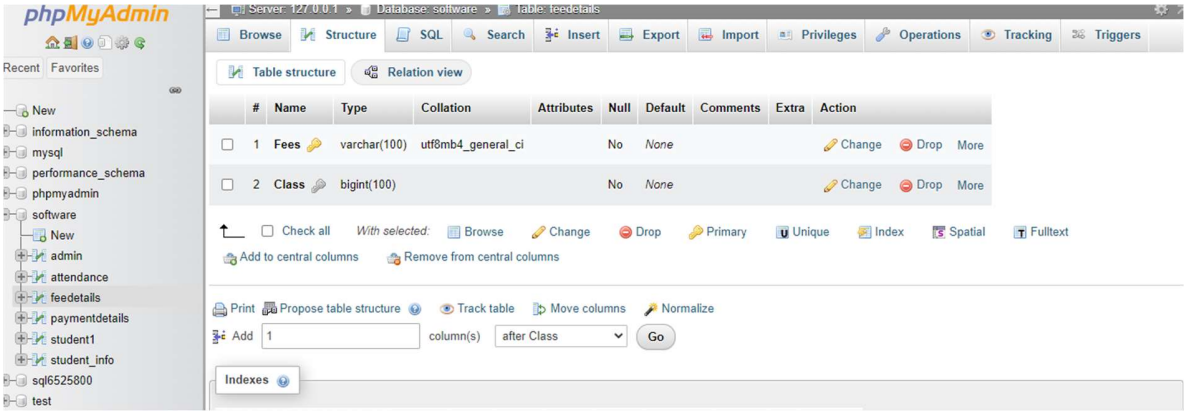
Structure of student credentials table



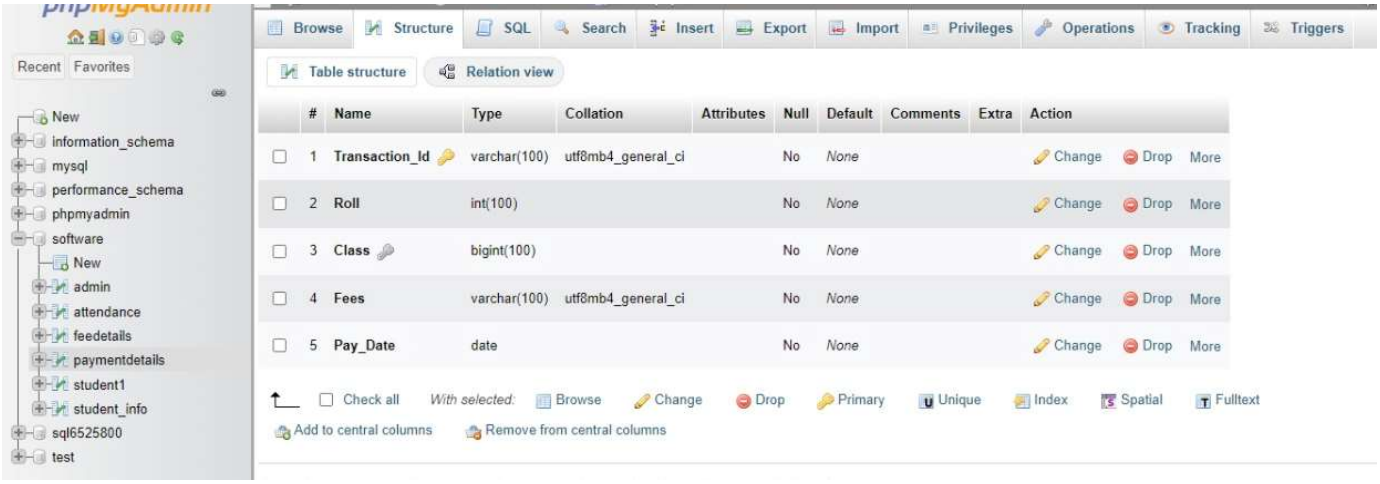
Attendance table



Structure of admin table

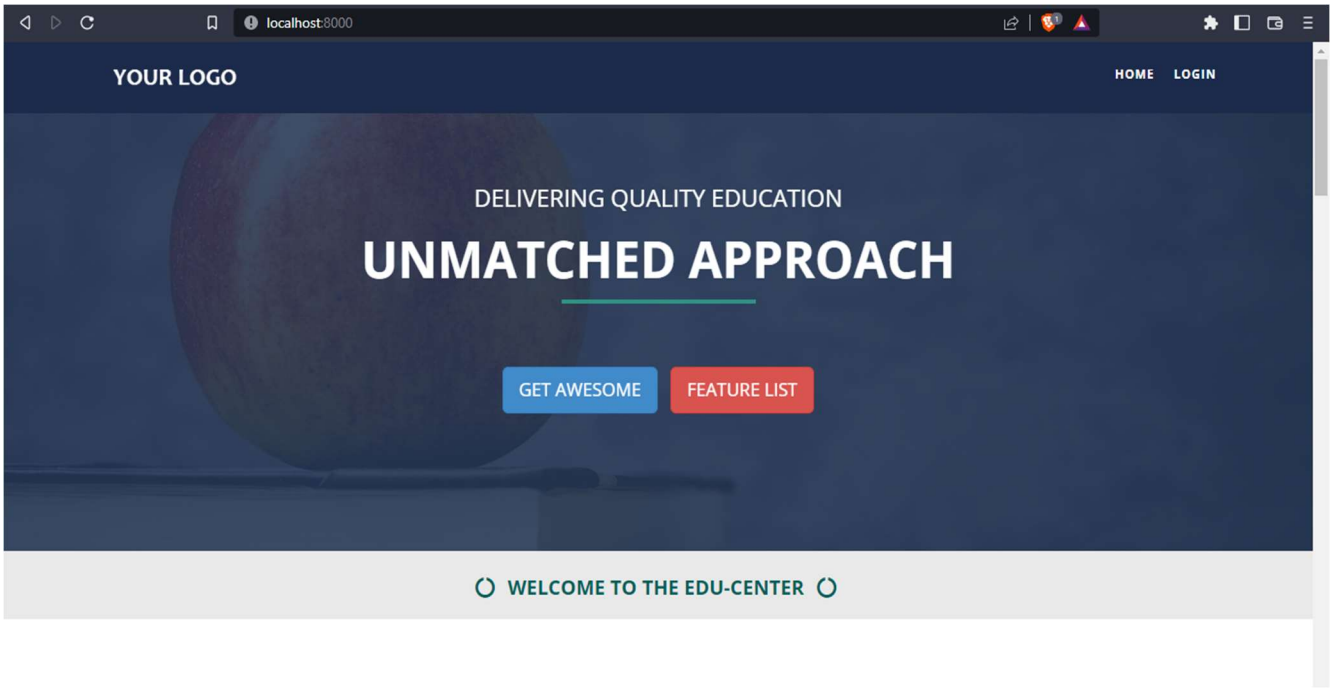


Structure of fee details table

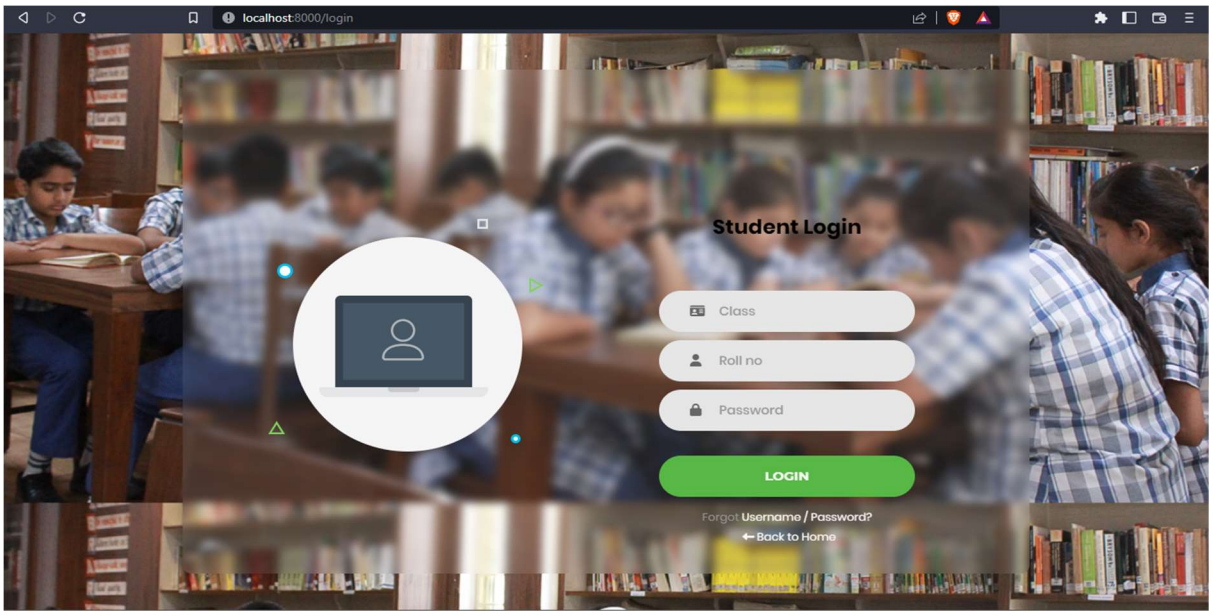


Structure of payment details table

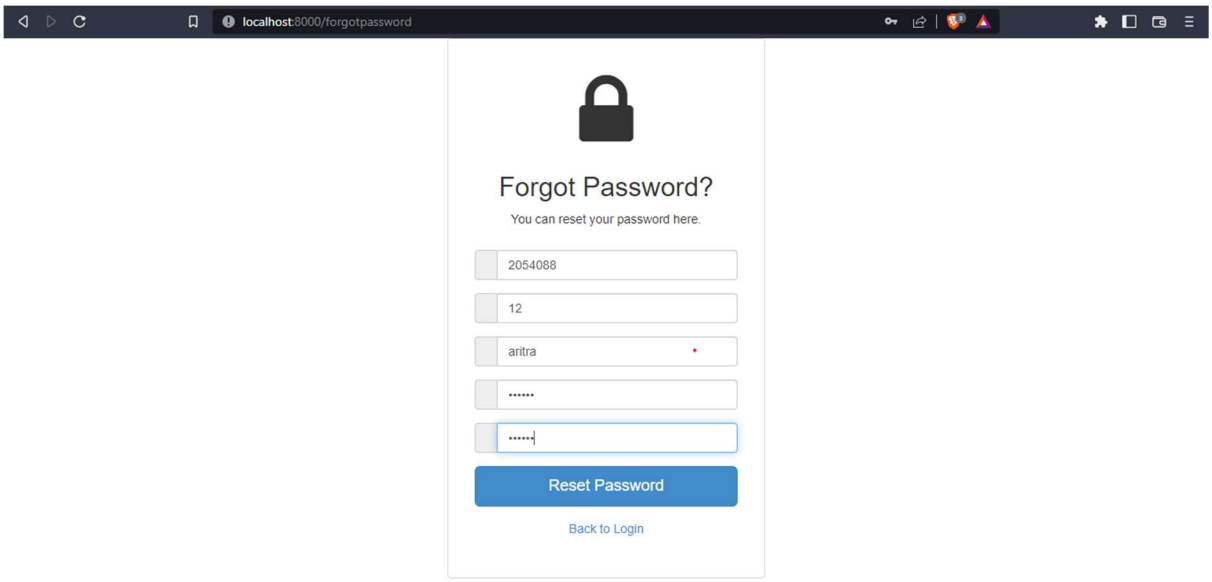
Screenshots of LIVE Mode Execution



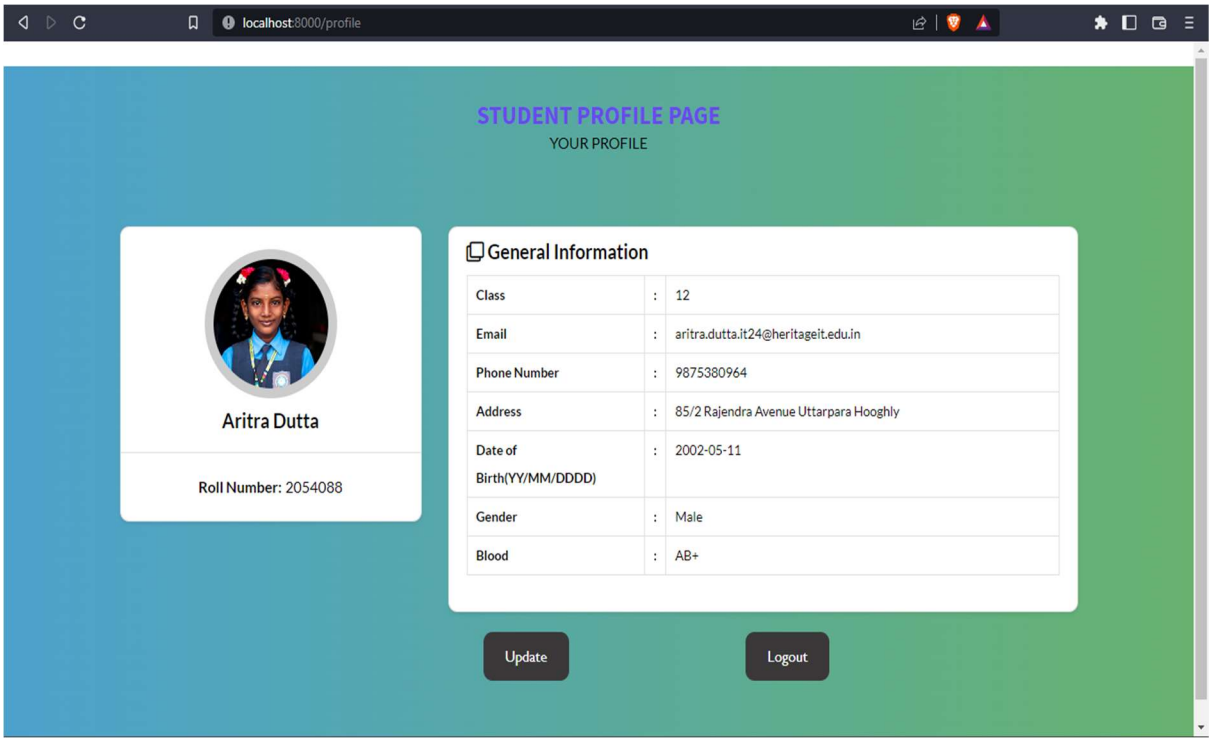
Landing page



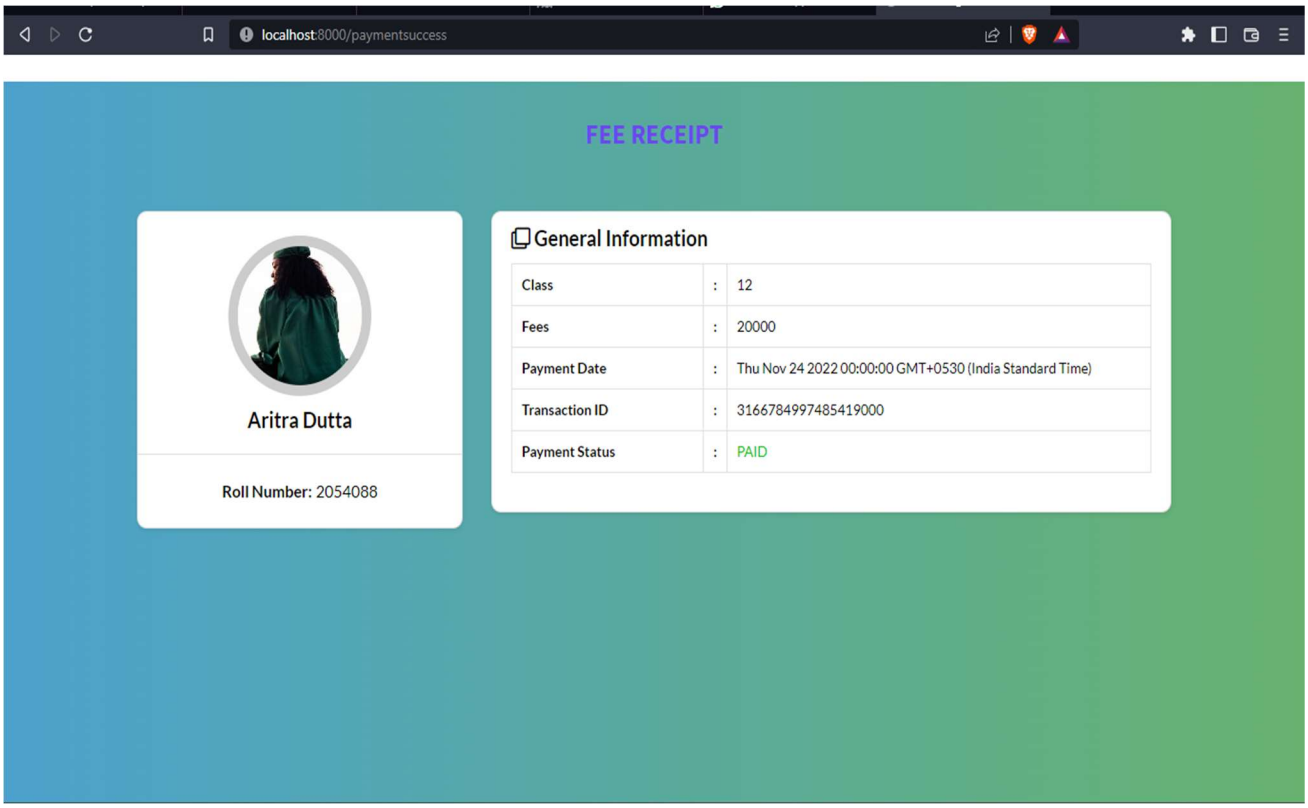
Login Page (Almost Same for Student and Admin)



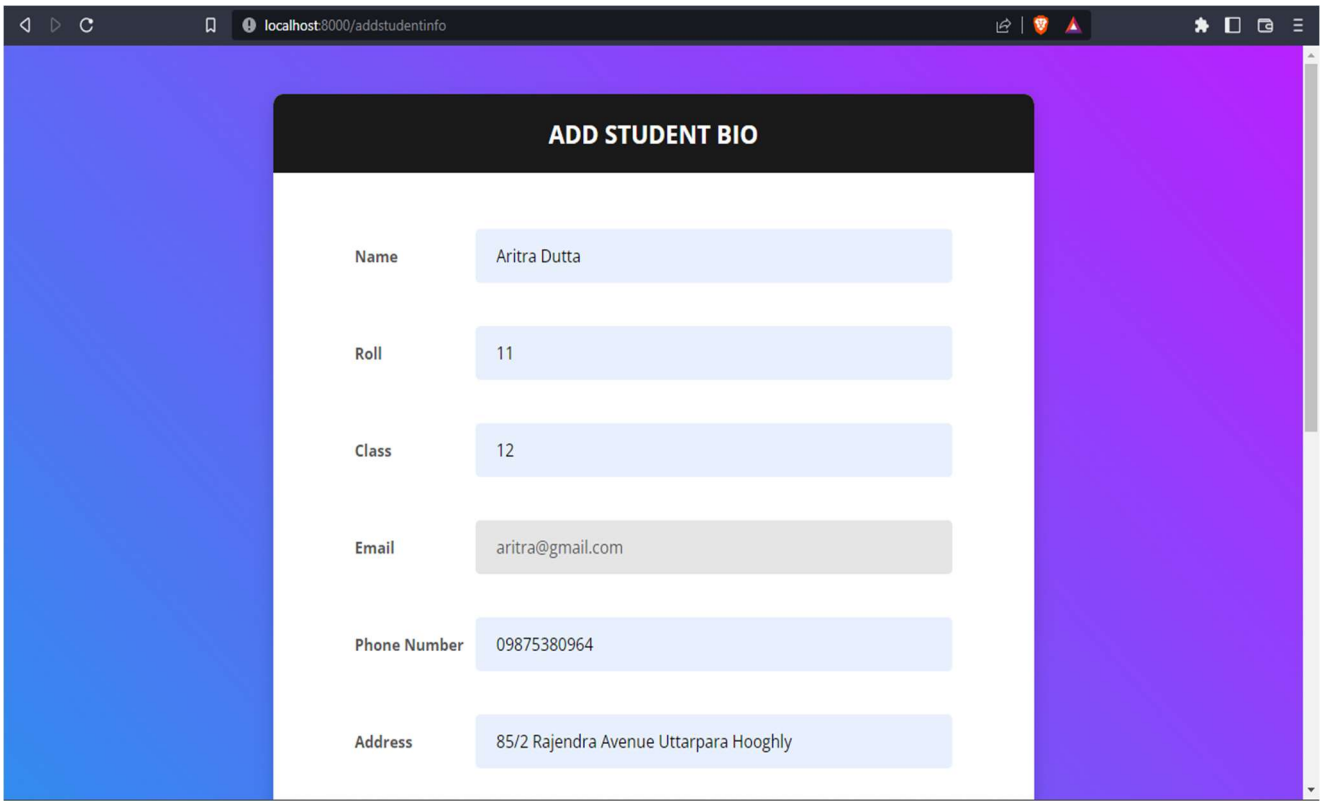
Forgot Password (Almost Same for Student and Admin)



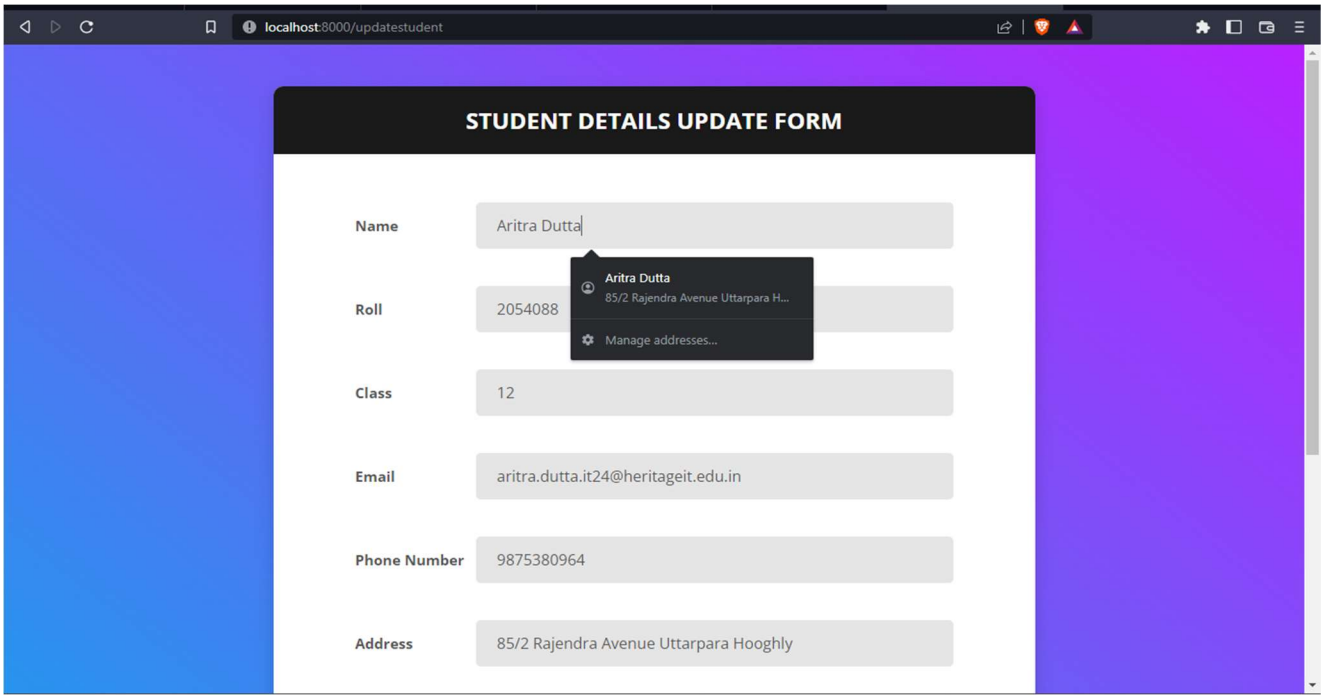
Profile Page (Almost Same for Student and Admin)



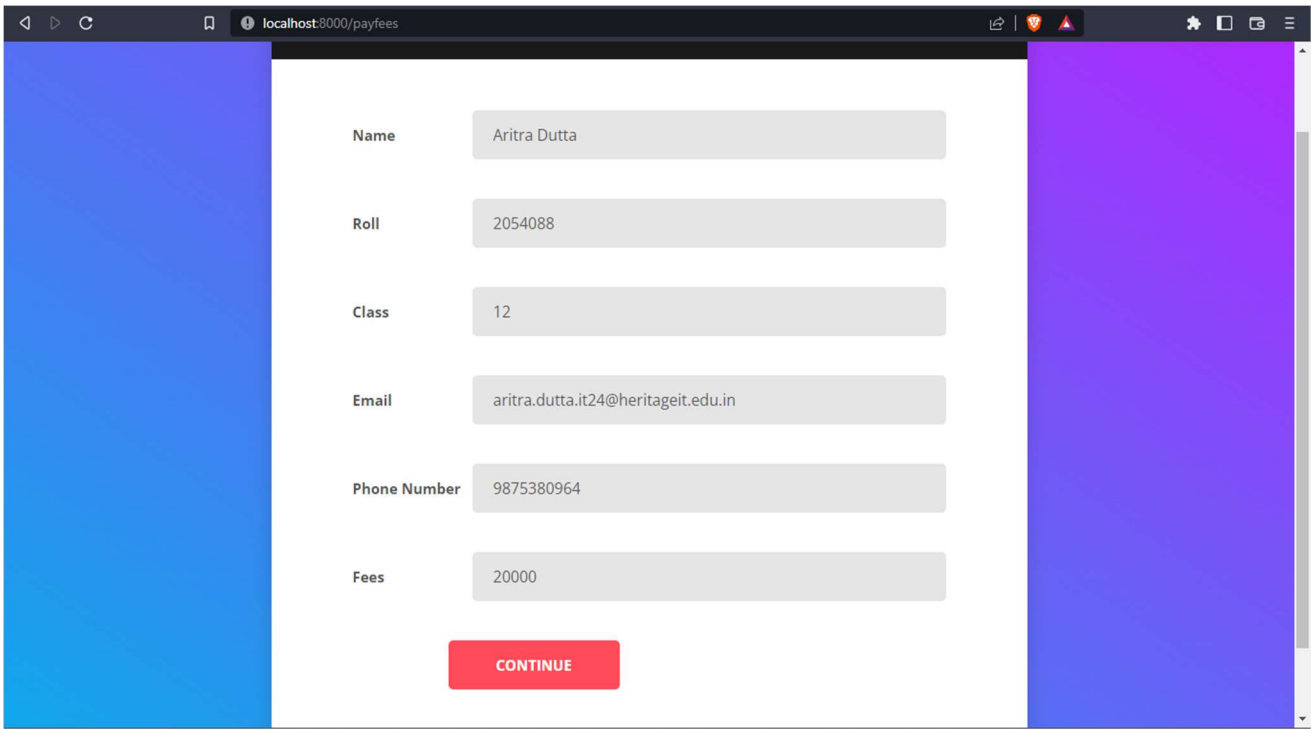
FEE RECEIPT PAGE



Add Student Page

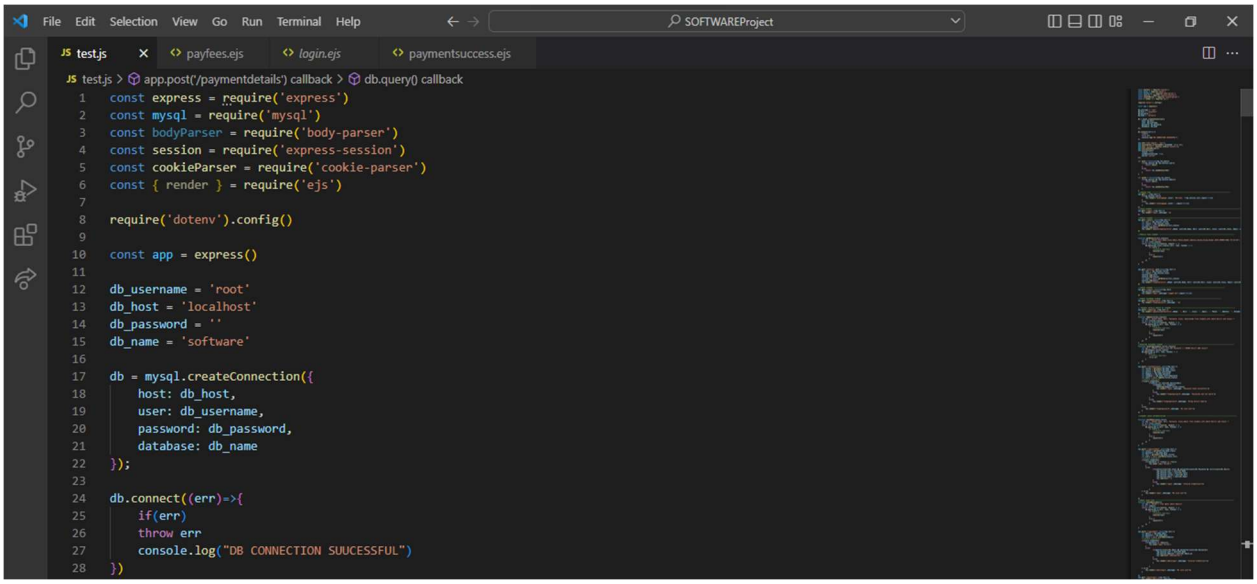


Update Student Page

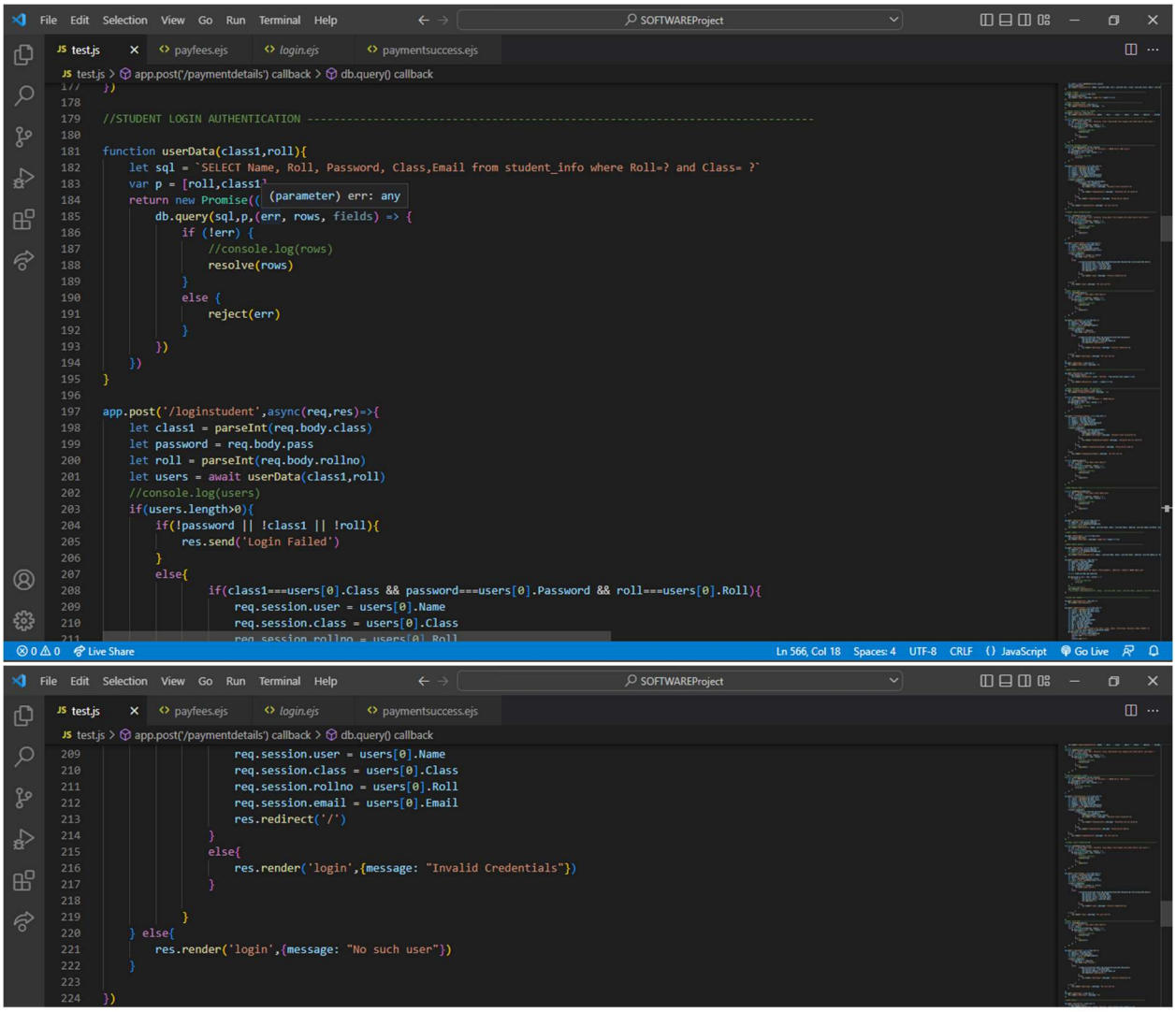


Payment Fees Page

Screenshot of important code segments

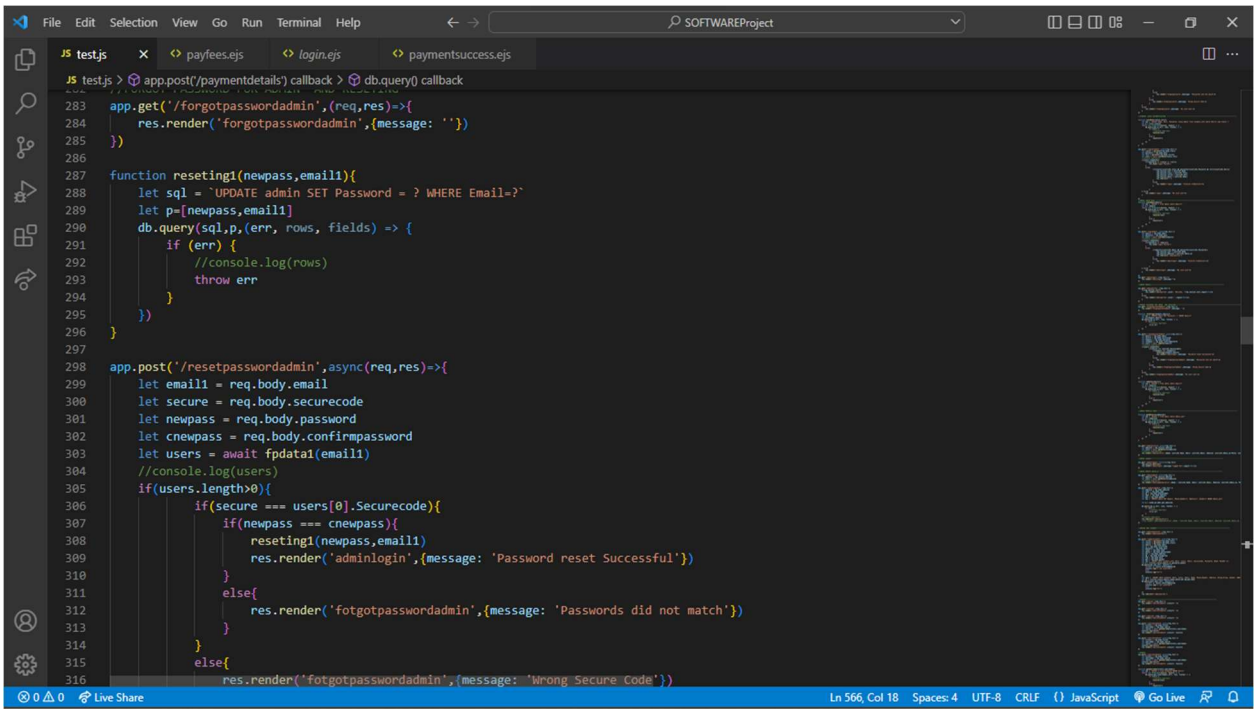


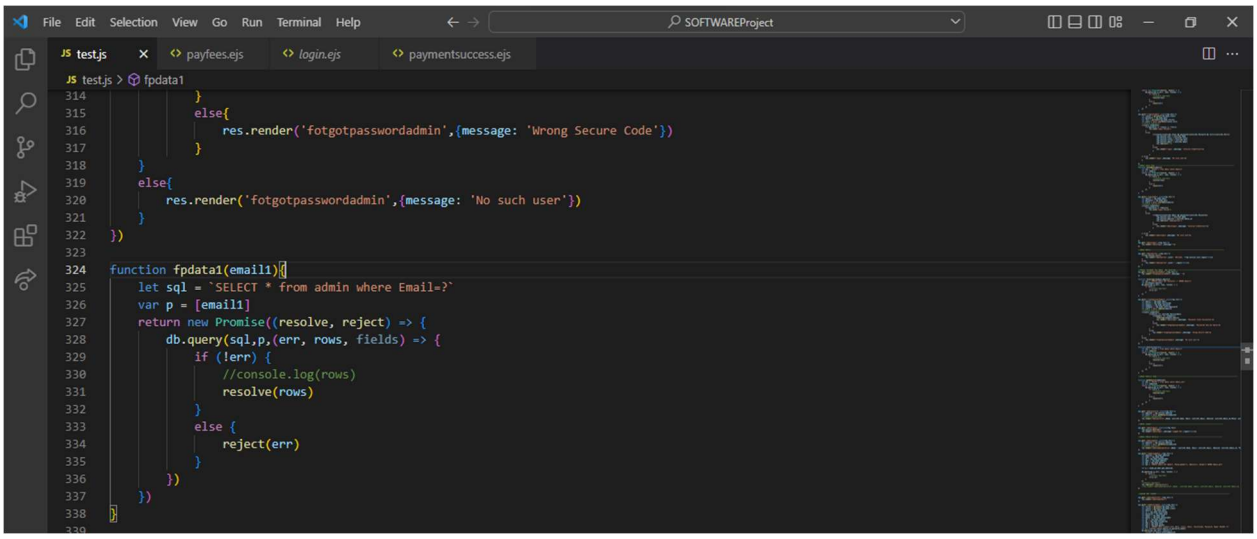
Required dipendencies & database connection



```
})

//STUDENT LOGIN AUTHENTICATION ------------------------------------------------------------------------

function userData(class1,roll){
    let sql = `SELECT Name, Roll, Password, Class,Email from student_info where Roll=? and Class= ?`
    var p = [roll,class1]
    return new Promise((  (parameter) err: any
        db.query(sql,p,(err, rows, fields) => {
            if (!err) {
                //console.log(rows)
                resolve(rows)
            }
            else {
                reject(err)
            }
        })
    })
}

app.post('/loginstudent',async(req,res)=>{
    let class1 = parseInt(req.body.class)
    let password = req.body.pass
    let roll = parseInt(req.body.rollno)
    let users = await userData(class1,roll)
    //console.log(users)
    if(users.length>0){
        if(!password || !class1 || !roll){
            res.send('Login Failed')
        }
        else{
                if(class1===users[0].Class && password===users[0].Password && roll===users[0].Roll){
                    req.session.user = users[0].Name
                    req.session.class = users[0].Class
                    req.session.rollno = users[0].Roll
                    req.session.email = users[0].Email
                    res.redirect('/')
                }
                else{
                    res.render('login',{message: "Invalid Credentials"})
                    }

                }
    } else{
        res.render('login',{message: "No such user"})
    }

})
```

Student login(almost same for admin)

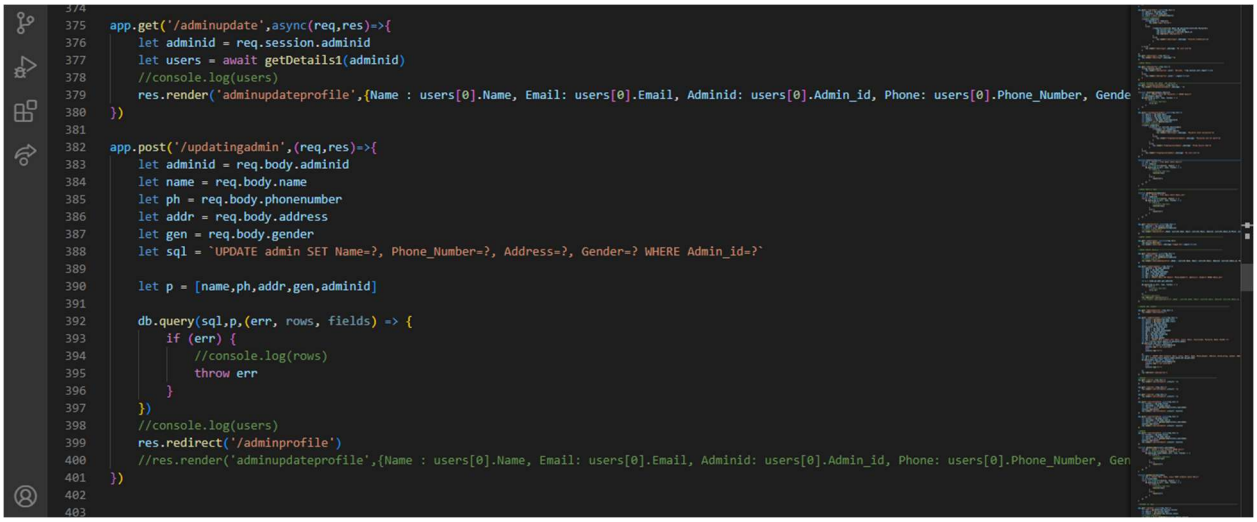




Forgot password(almost same for student and admin)

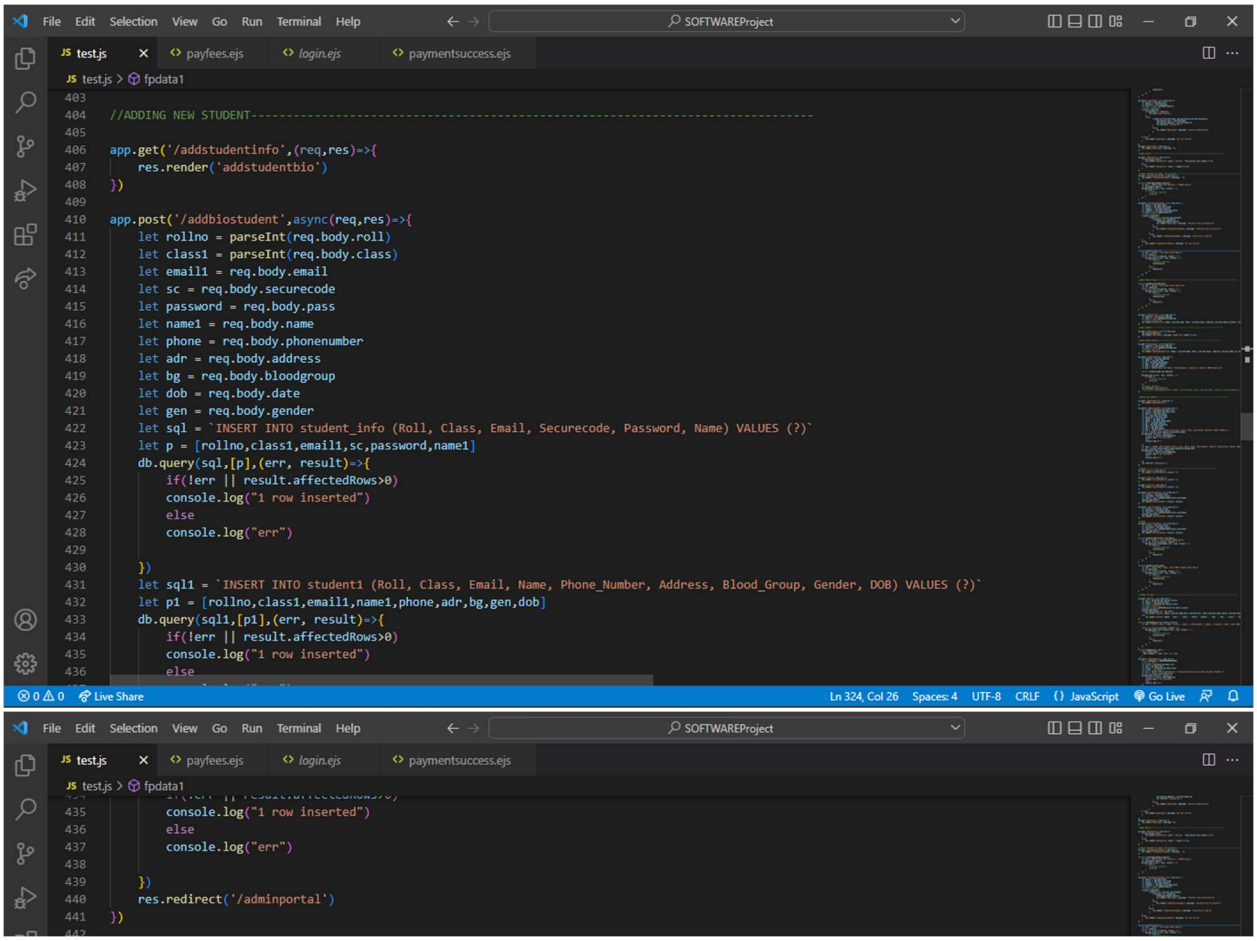
```
function getDetails1(adminid){
    let sql = `SELECT * from admin where Admin_id=?`
    var p = [adminid]
    return new Promise((resolve, reject) => {
        db.query(sql,p,(err, rows, fields) => {
            if (!err) {
                //console.log(rows)
                resolve(rows)
            }
            else {
                reject(err)
            }
        })
    })
}


app.get('/adminprofile',async(req,res)=>{
    let adminid = req.session.adminid
    let users = await getDetails1(adminid)
    //console.log(adminid)
    res.render('adminprofile',{Name: users[0].Name, Email: users[0].Email, Adminid: users[0].Admin_id,Phone: users[0].Phone_Number, Address: user
})
```

Profile page(almost same for student and admin)



```
app.get('/adminupdate',async(req,res)=>{
    let adminid = req.session.adminid
    let users = await getDetails1(adminid)
    //console.log(users)
    res.render('adminupdateprofile',{Name : users[0].Name, Email: users[0].Email, Adminid: users[0].Admin_id, Phone: users[0].Phone_Number, Gende
})

app.post('/updatingadmin',(req,res)=>{
    let adminid = req.body.adminid
    let name = req.body.name
    let ph = req.body.phonenumber
    let addr = req.body.address
    let gen = req.body.gender
    let sql = `UPDATE admin SET Name=?, Phone_Number=?, Address=?, Gender=? WHERE Admin_id=?`

    let p = [name,ph,addr,gen,adminid]

    db.query(sql,p,(err, rows, fields) => {
        if (err) {
            //console.log(rows)
            throw err
        }
    })
    //console.log(users)
    res.redirect('/adminprofile')
    //res.render('adminupdateprofile',{Name : users[0].Name, Email: users[0].Email, Adminid: users[0].Admin_id, Phone: users[0].Phone_Number, Gen
})
```

Updating details(almost same for student and admin)

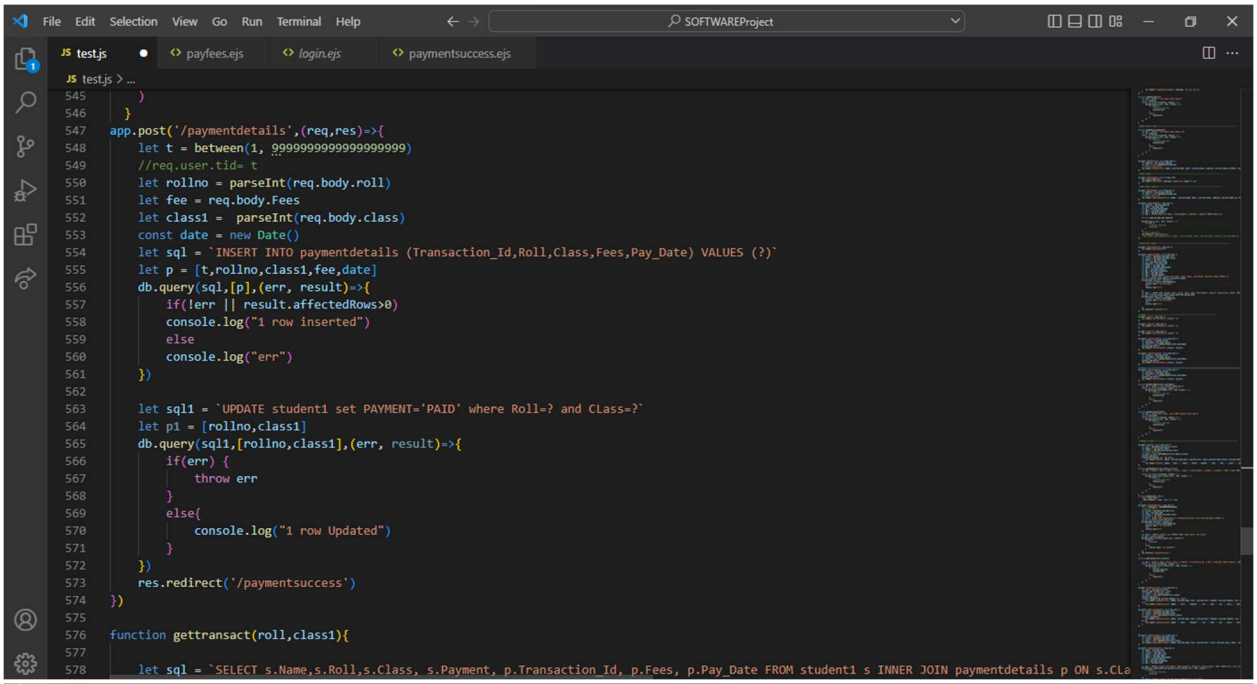


```
//ADDING NEW STUDENT-----------------------------------------------------------------------------

app.get('/addstudentinfo',(req,res)=>{
    res.render('addstudentbio')
})

app.post('/addbiostudent',async(req,res)=>{
    let rollno = parseInt(req.body.roll)
    let class1 = parseInt(req.body.class)
    let email1 = req.body.email
    let sc = req.body.securecode
    let password = req.body.pass
    let name1 = req.body.name
    let phone = req.body.phonenumber
    let adr = req.body.address
    let bg = req.body.bloodgroup
    let dob = req.body.date
    let gen = req.body.gender
    let sql = `INSERT INTO student_info (Roll, Class, Email, Securecode, Password, Name) VALUES (?)`
    let p = [rollno,class1,email1,sc,password,name1]
    db.query(sql,[p],(err, result)=>{
        if(!err || result.affectedRows>0)
        console.log("1 row inserted")
        else
        console.log("err")

    })
    let sql1 = `INSERT INTO student1 (Roll, Class, Email, Name, Phone_Number, Address, Blood_Group, Gender, DOB) VALUES (?)`
    let p1 = [rollno,class1,email1,name1,phone,adr,bg,gen,dob]
    db.query(sql1,[p1],(err, result)=>{
        if(!err || result.affectedRows>0)
        console.log("1 row inserted")
        else
        console.log("err")

    })
    res.redirect('/adminportal')
})
```

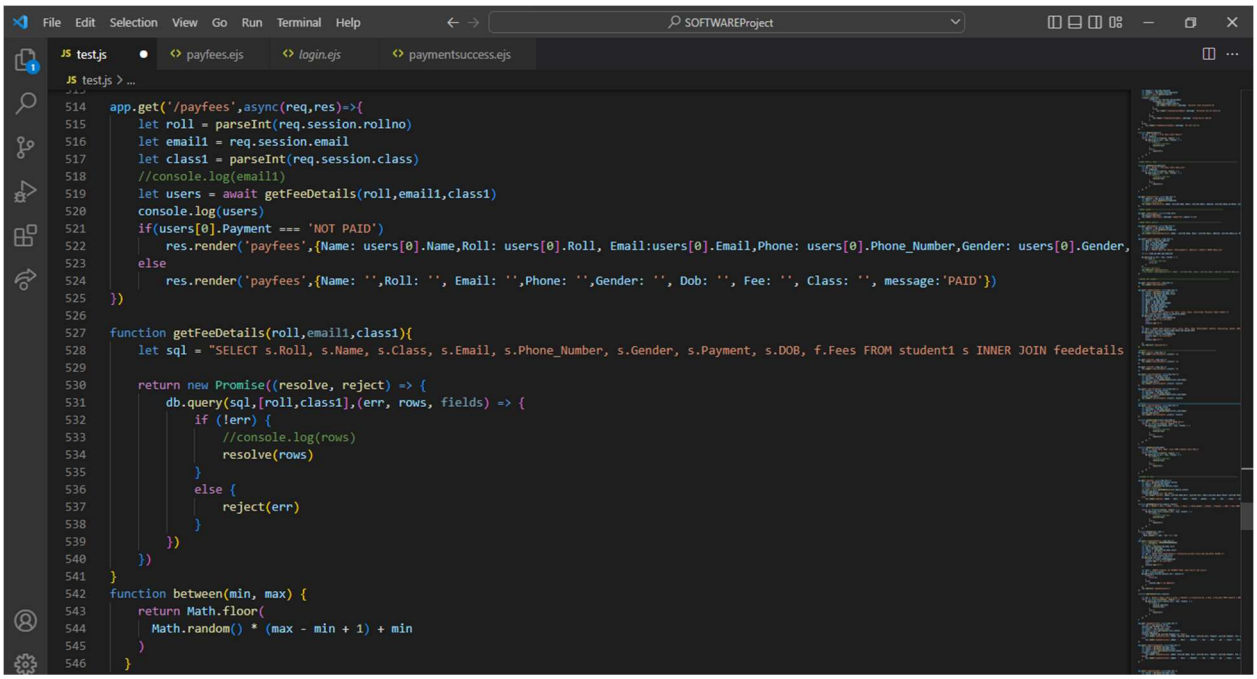
Adding students

```
app.post('/searchstudent1',async(req,res)=>{
    let filter1 = req.body.filter
    let searched1 = req.body.search
    let result = await getSearched(filter1,searched1)
    console.log(result)
    res.render('searchstudent1',{result: result})
})

function getSearched(filter1,searched1){
    let sql = `SELECT * from student1 WHERE Roll=?`
    return new Promise((resolve, reject) => {
        db.query(sql,[searched1],(err, rows, fields) => {
            if (!err) {
                //console.log(rows)
                resolve(rows)
            }
            else {
                reject(err)
            }
        })
    })
}
```

Searching students

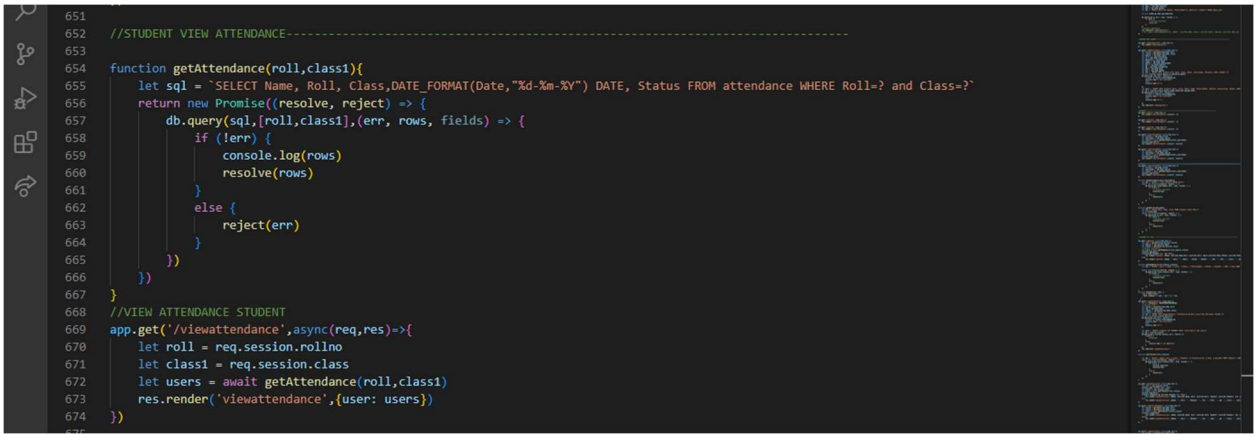

File Edit Selection View Go Run Terminal Help
SOFTWAREProject
test.js
payfees.ejs
login.ejs
paymentsuccess.ejs
test.js > ...
)
}
app.post('/paymentdetails',(req,res)=>{
let t = between(1, 9999999999999999999)
//req.user.tid= t
let rollno = parseInt(req.body.roll)
let fee = req.body.Fees
let class1 = parseInt(req.body.class)
const date = new Date()
let sql = `INSERT INTO paymentdetails (Transaction_Id,Roll,Class,Fees,Pay_Date) VALUES (?)`
let p = [t,rollno,class1,fee,date]
db.query(sql,[p],(err, result)=>{
if(!err || result.affectedRows>0)
console.log("1 row inserted")
else
console.log("err")
})
let sql1 = `UPDATE student1 set PAYMENT='PAID' where Roll=? and CLass=?`
let p1 = [rollno,class1]
db.query(sql1,[rollno,class1],(err, result)=>{
if(err) {
throw err
}
else{
console.log("1 row Updated")
}
})
res.redirect('/paymentsuccess')
})
function gettransact(roll,class1){
let sql = `SELECT s.Name,s.Roll,s.Class, s.Payment, p.Transaction_Id, p.Fees, p.Pay_Date FROM student1 s INNER JOIN paymentdetails p ON s.CLa


function gettransact(roll,class1){
let sql = `SELECT s.Name,s.Roll,s.Class, s.Payment, p.Transaction_Id, p.Fees, p.Pay_Date FROM student1 s INNER JOIN paymentdetails p ON s.CLa
return new Promise((resolve, reject) => {
db.query(sql,[roll,class1],(err, rows, fields) => {
if (!err) {
console.log(rows)
resolve(rows)
}
else {
reject(err)
}
})
})
}
app.get('/paymentsuccess',async(req,res)=>{
let roll = req.session.rollno
console.log('PAYMENT ROLL',roll)
let class1 = req.session.class
let users = await gettransact(roll,class1)
console.log(users)
if(users.length>0 && users[0].Payment === 'PAID')
res.render('paymentsuccess',{Name: users[0].Name, Roll: users[0].Roll, Payment: users[0].Payment, Tid: users[0].Transaction_Id, Fees: use
else
res.render('paymentsuccess',{Name: '', Roll: '', Payment: '', Tid: '', Fees: '', pd: '', Class: '', message:'NOT YET PAID'})
})



```
app.get('/payfees',async(req,res)=>{
    let roll = parseInt(req.session.rollno)
    let email1 = req.session.email
    let class1 = parseInt(req.session.class)
    //console.log(email1)
    let users = await getFeeDetails(roll,email1,class1)
    console.log(users)
    if(users[0].Payment === 'NOT PAID')
        res.render('payfees',{Name: users[0].Name,Roll: users[0].Roll, Email:users[0].Email,Phone: users[0].Phone_Number,Gender: users[0].Gender,
    else
        res.render('payfees',{Name: '',Roll: '', Email: '',Phone: '',Gender: '', Dob: '', Fee: '', Class: '', message:'PAID'})
})

function getFeeDetails(roll,email1,class1){
    let sql = "SELECT s.Roll, s.Name, s.Class, s.Email, s.Phone_Number, s.Gender, s.Payment, s.DOB, f.Fees FROM student1 s INNER JOIN feedetails

    return new Promise((resolve, reject) => {
        db.query(sql,[roll,class1],(err, rows, fields) => {
            if (!err) {
                //console.log(rows)
                resolve(rows)
            }
            else {
                reject(err)
            }
        })
    })
}
function between(min, max) {
    return Math.floor(
      Math.random() * (max - min + 1) + min
    )
  }
```

Fees payment



```
//STUDENT VIEW ATTENDANCE-----------------------------------------------------------------------------

function getAttendance(roll,class1){
    let sql = `SELECT Name, Roll, Class,DATE_FORMAT(Date,"%d-%m-%Y") DATE, Status FROM attendance WHERE Roll=? and Class=?`
    return new Promise((resolve, reject) => {
        db.query(sql,[roll,class1],(err, rows, fields) => {
            if (!err) {
                console.log(rows)
                resolve(rows)
            }
            else {
                reject(err)
            }
        })
    })
}
//VIEW ATTENDANCE STUDENT
app.get('/viewattendance',async(req,res)=>{
    let roll = req.session.rollno
    let class1 = req.session.class
    let users = await getAttendance(roll,class1)
    res.render('viewattendance',{user: users})
})
```

Student view attendance

BIBLIOGRAPHY

BOOKS/REFFERENCES	AUTHORS/SOURCE
1. Software engineering	Sajan Mathew
2. INTERNET	
○ COLORIB	https://colorlib.com/
○ W3C	http://www.w3.org/
○ Free CSS	https://www.free-css.com/free-css-templates
○ Font Awesome	https://fontawesome.com/