

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

ACKNOWLEDGEMENT

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
25/11/22

Aritrick Bhowmick
25/11/22

Sohini Ghosh
25/11/22

Aritra Dutta

Aritrick Bhowmick

Sohini Ghosh

INDEX

SL NO	CONTENT	PAGE NO
1	Hardware & Software requirements	1
2	System Requirement Specifications <ul style="list-style-type: none"> • Introduction <ul style="list-style-type: none"> ◦ Purpose ◦ Project scope ◦ References • System Description • System Features • Database Dictionary • External Interface Requirements • Technical Requirements (Non functional) • Open Issues 	2-8
3	Entity relationship diagram	9
4	Data flow diagram <ul style="list-style-type: none"> • Context level (0 level) • 1st level • 2nd level 	10-13
5	UML Diagram <ul style="list-style-type: none"> • Class diagram • Use case diagram • Sequence diagram • Activity diagram 	14 15 16-17 18

7	Database design screenshots	19-22
8	Screenshot of live mode execution	23-26
9	Screenshot of important code	27-33
10	Bibliography	34

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.

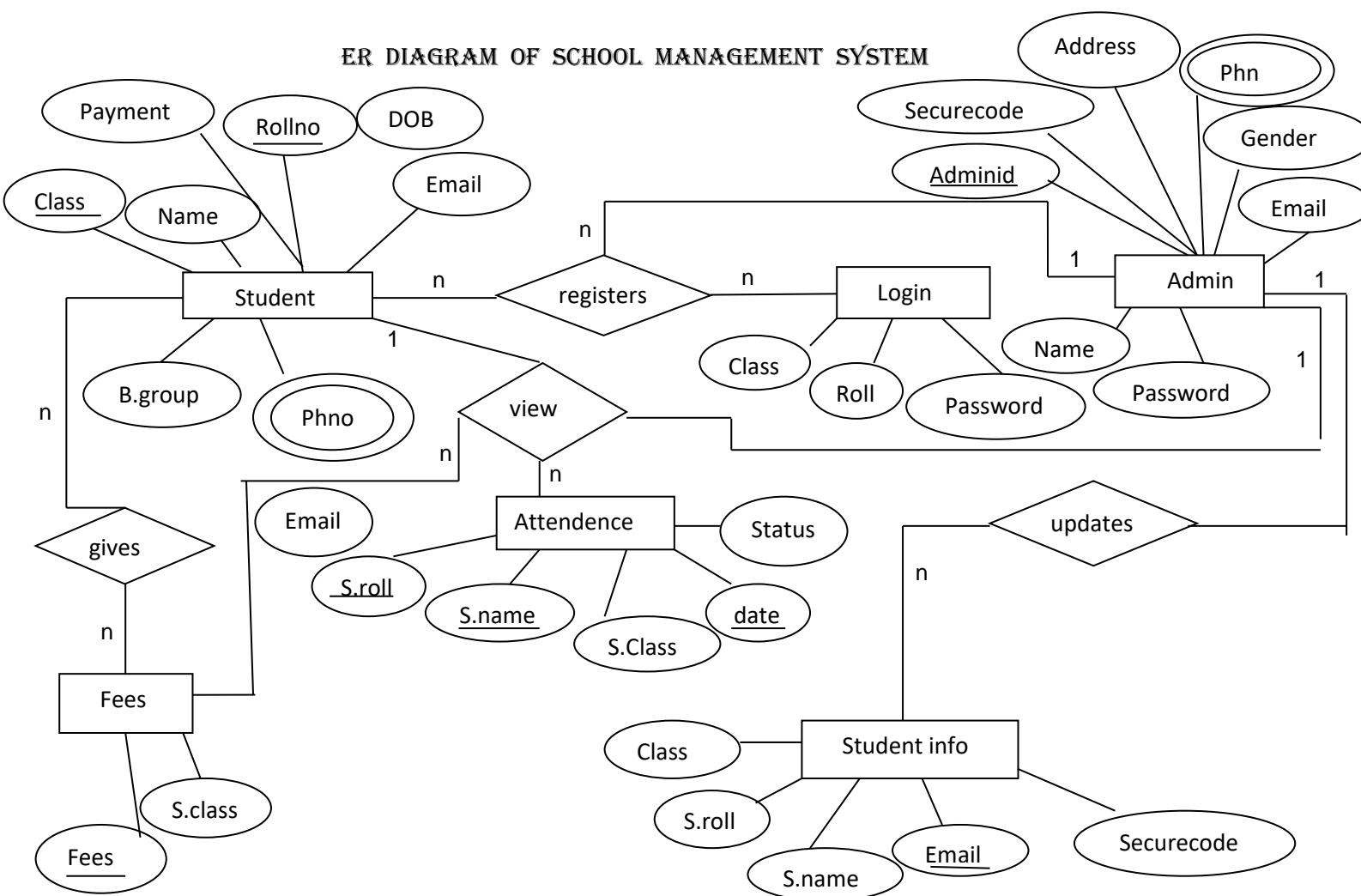
Maintainability

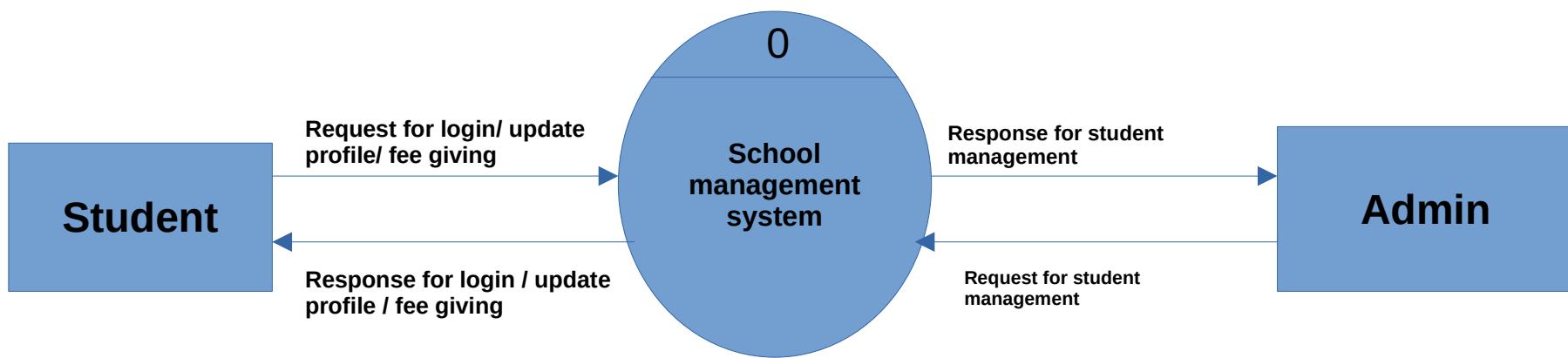
Easy to maintain and easily updatable.

Open Issues

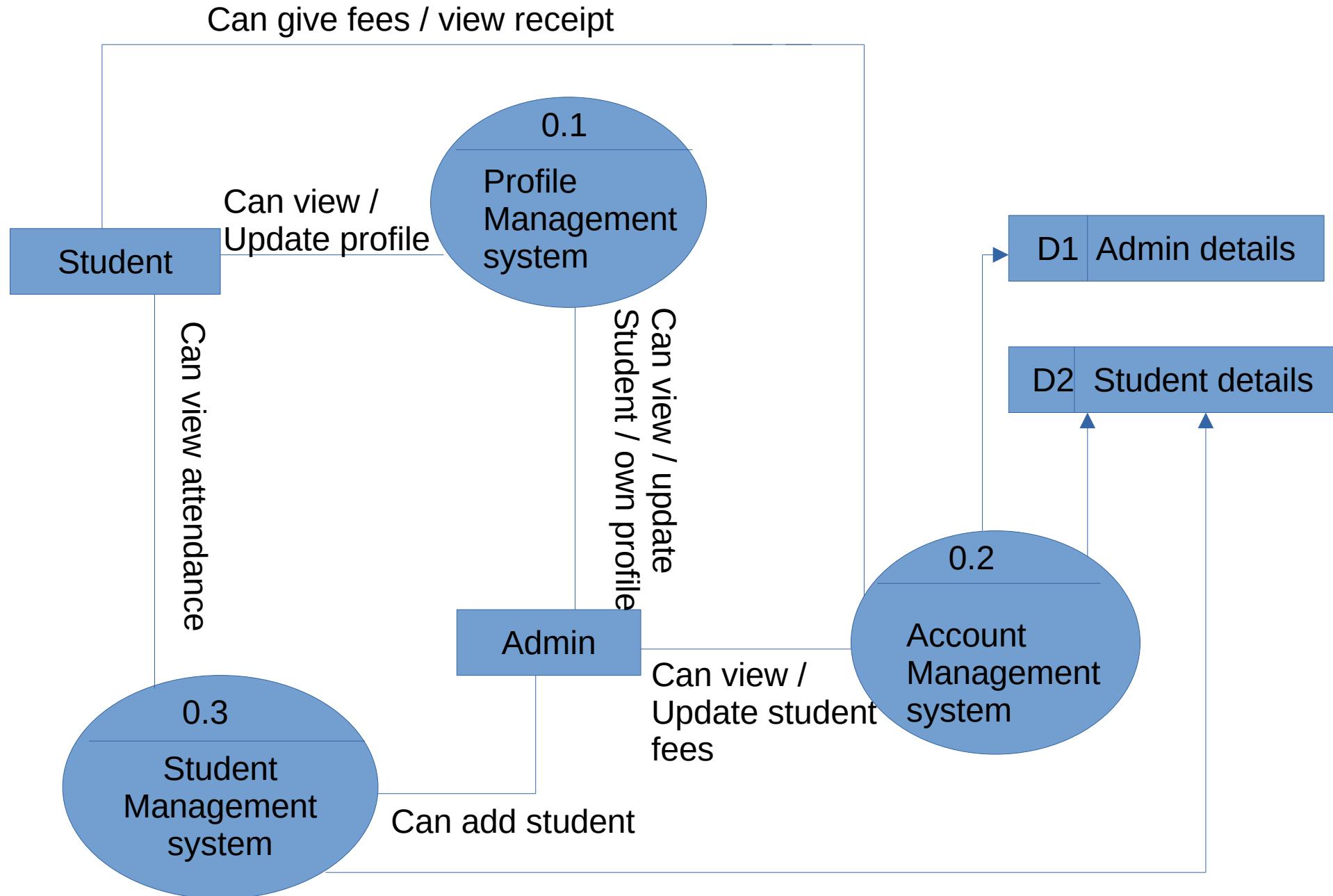
Frontend could be more attractive and there could be an option of Faculty management and Student Management System can be made more efficient

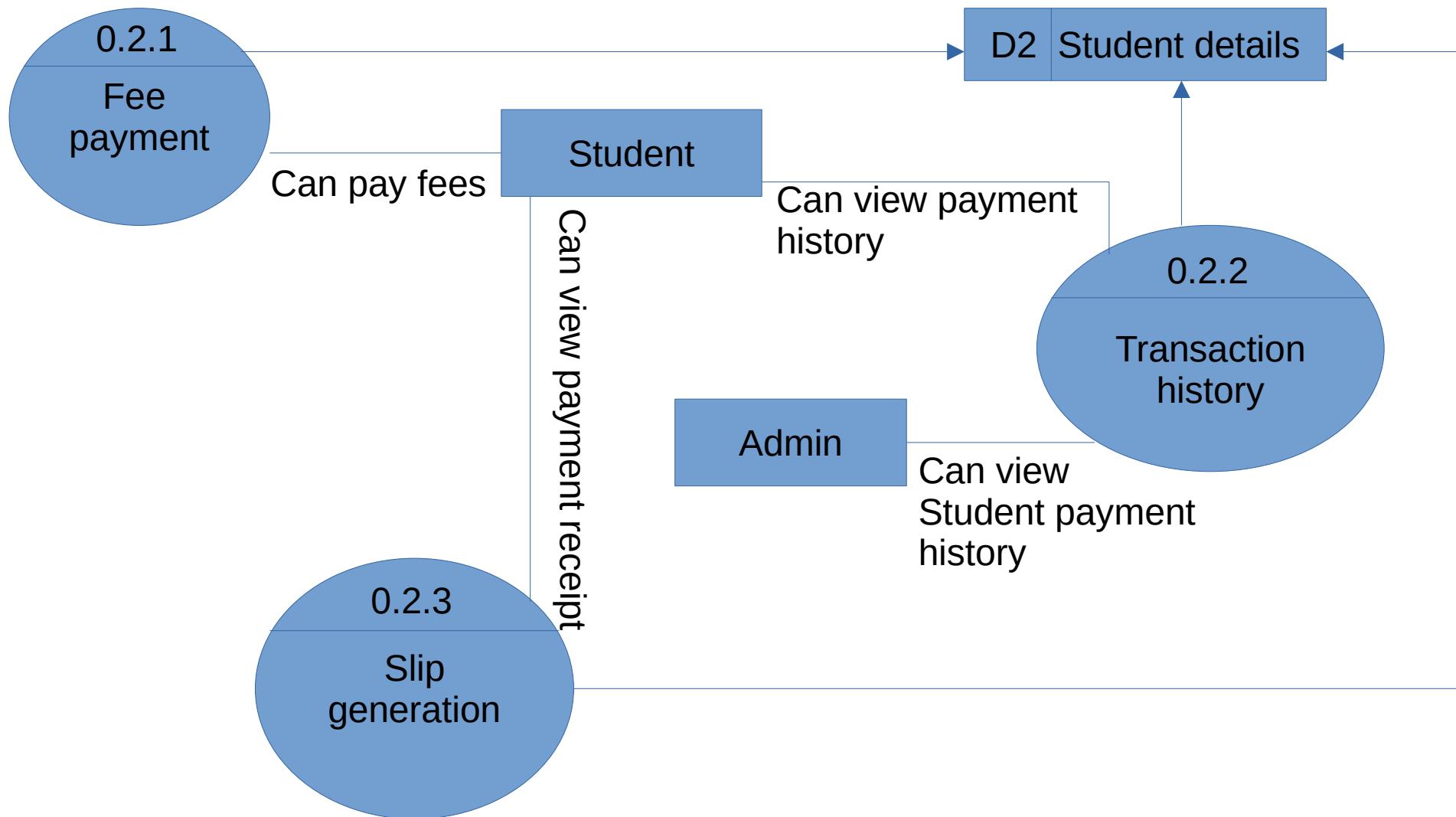
ER DIAGRAM OF SCHOOL MANAGEMENT SYSTEM



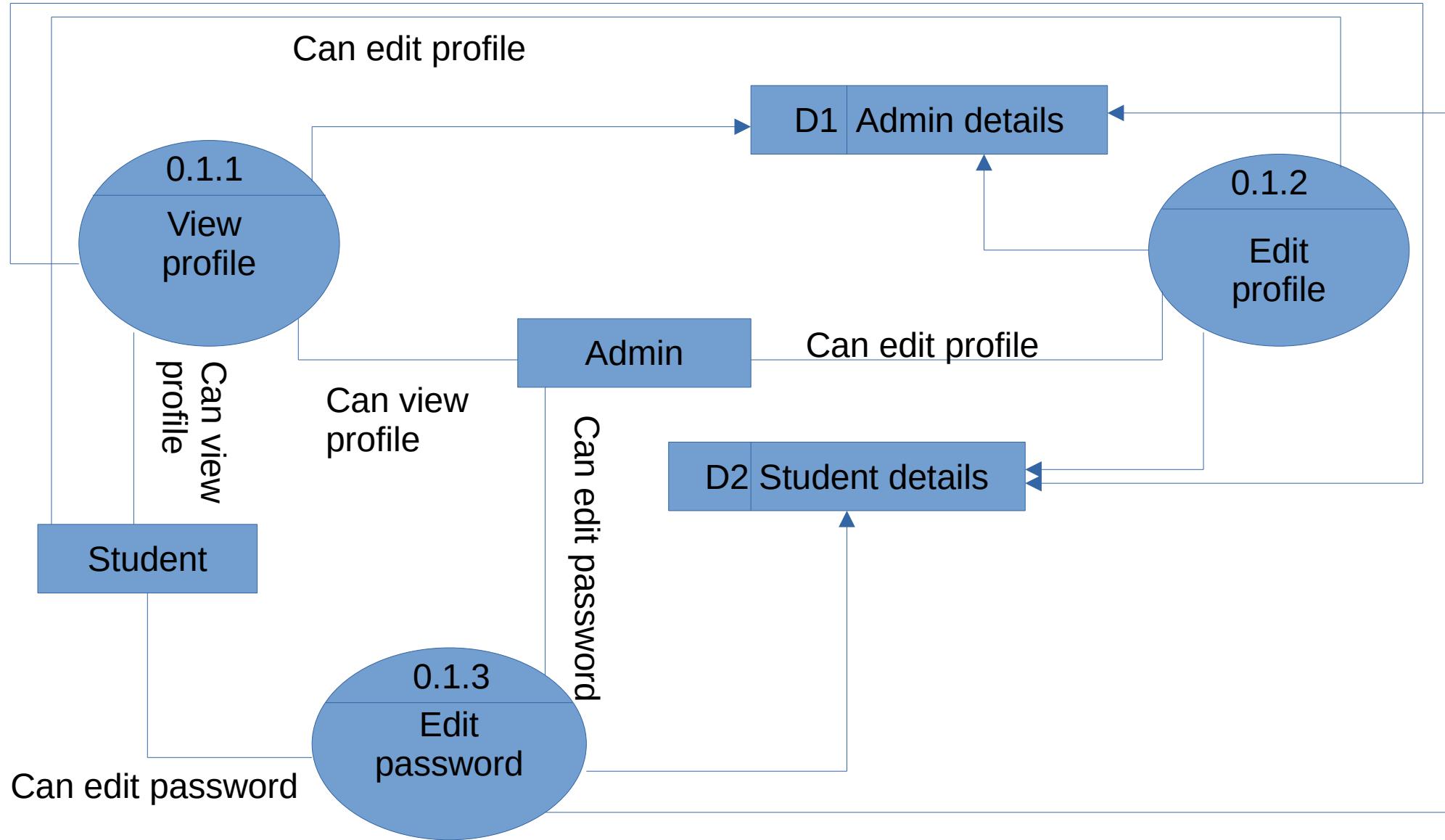


Level 0 DFD

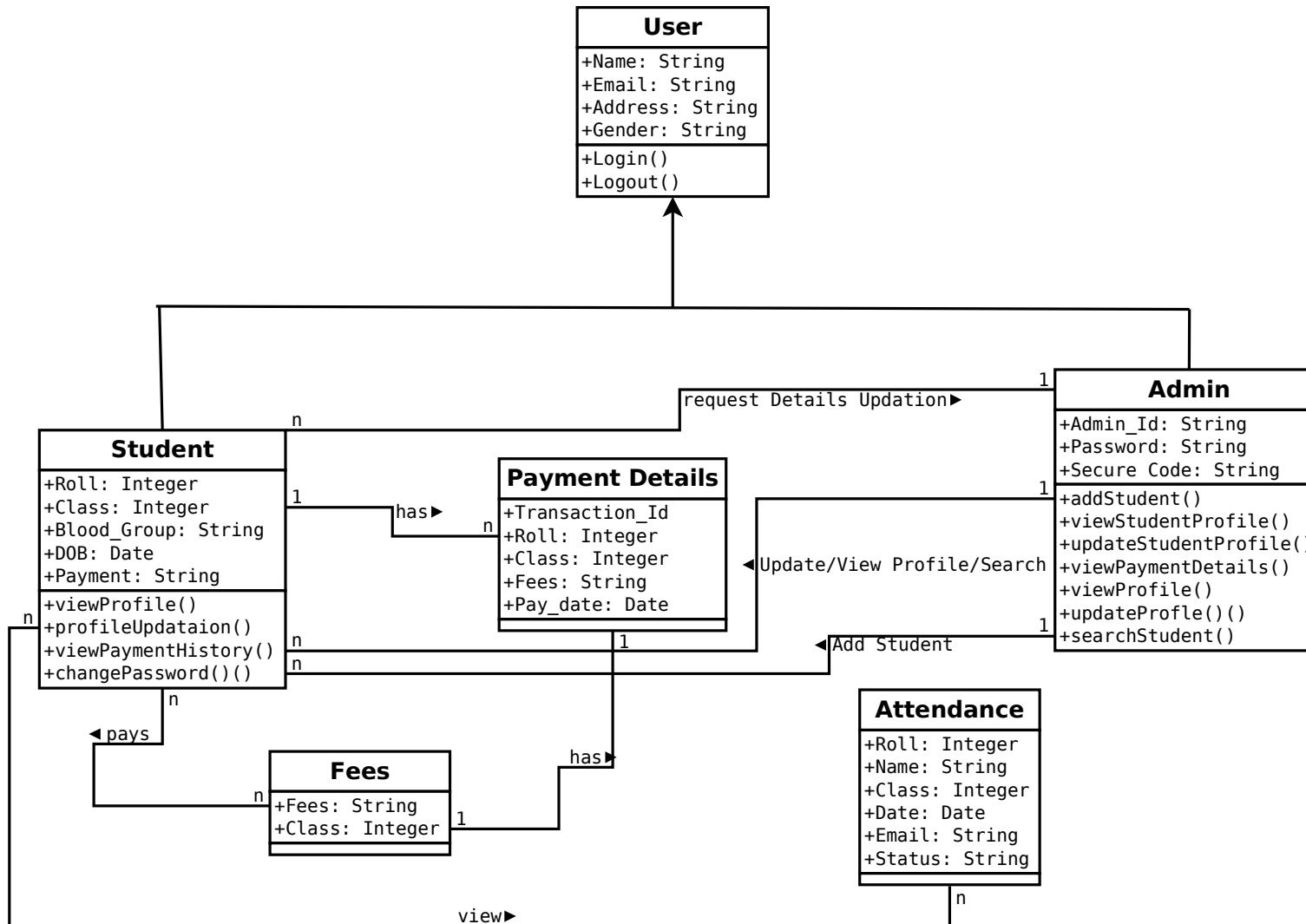




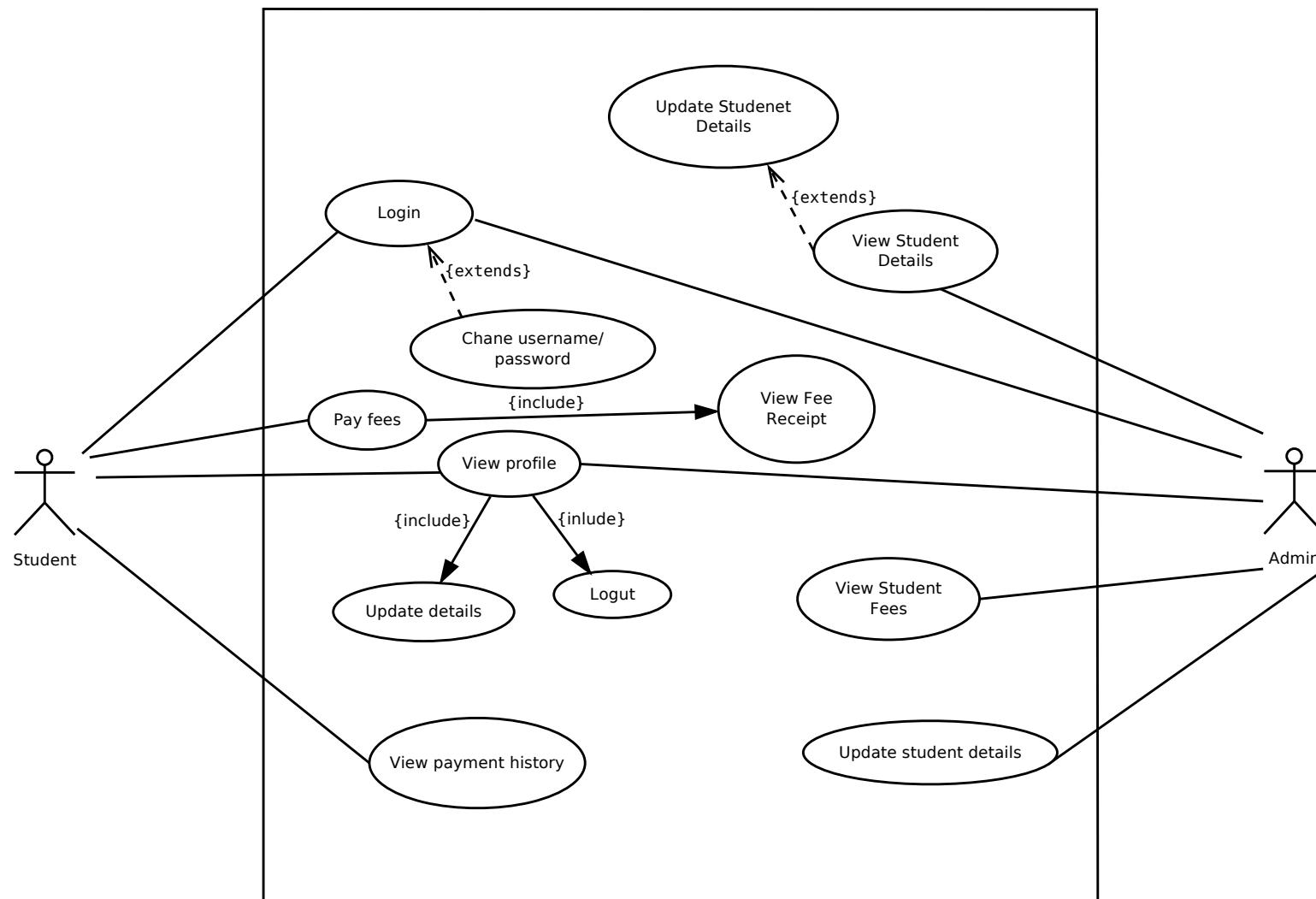
2 level DFD of school management system



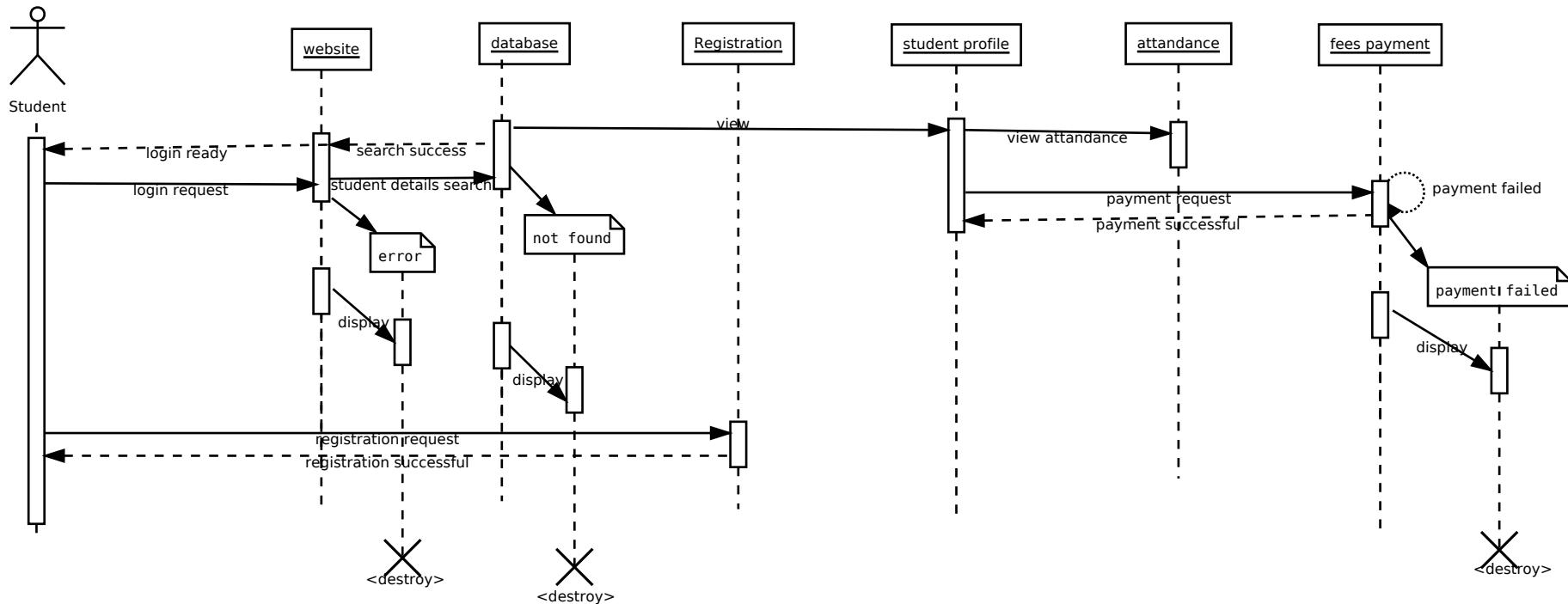
CLASS DIAGRAM



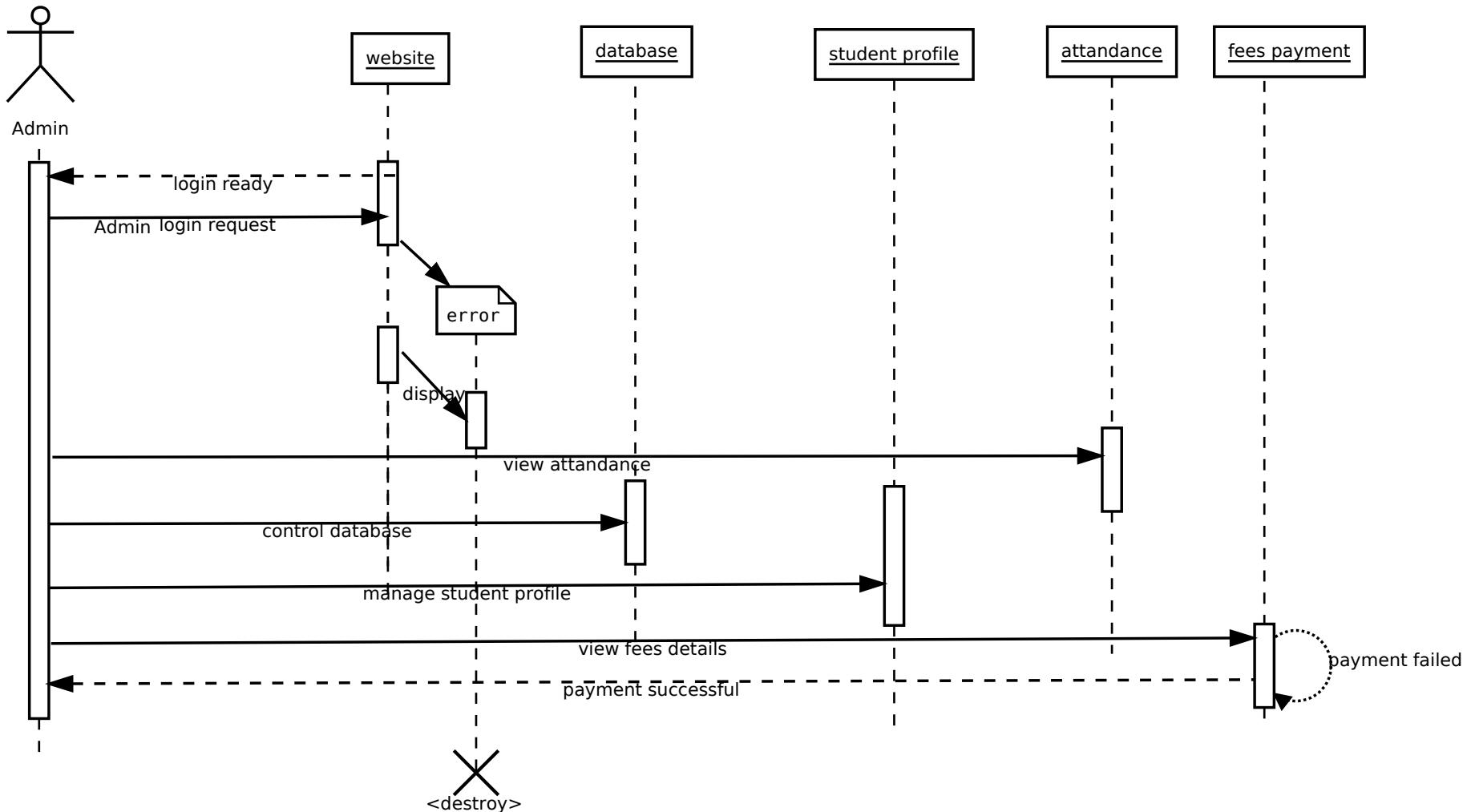
USE CASE DIAGRAM



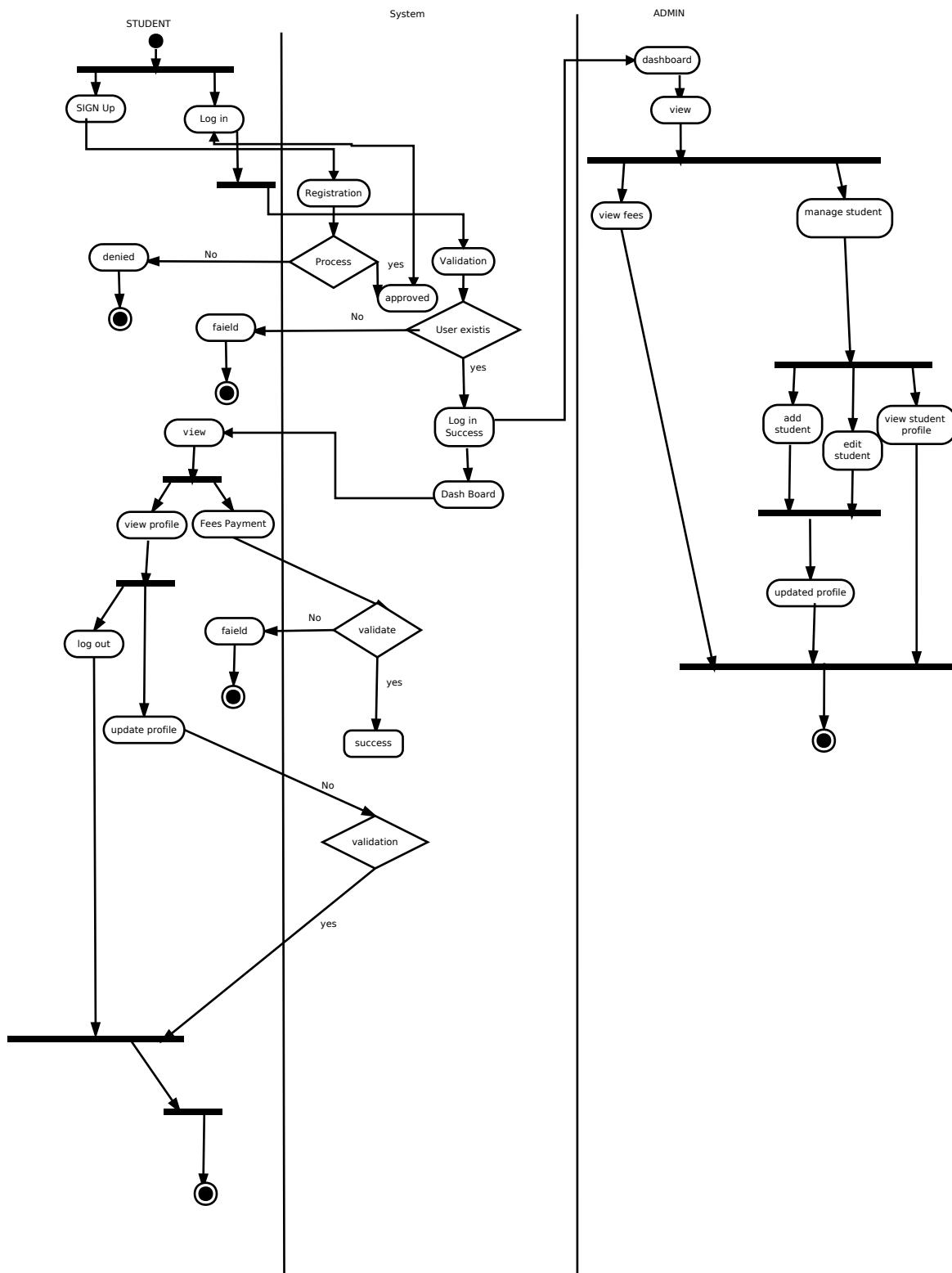
SEQUENCE DIAGRAM FOR STUDENT



SEQUENCE DIAGRAM FOR ADMIN



ACTIVITY DIAGRAM



Database Design

The screenshot shows the phpMyAdmin interface for the 'software' database. The left sidebar lists databases: New, information_schema, mysql, performance_schema, phpmyadmin, software, sql6525800, and test. The 'software' database is selected. The main area displays the following table structure:

Table	Action	Rows	Type	Collation	Size	Overhead
admin	Browse Structure Search Insert Empty Drop	1	InnoDB	utf8mb4_general_ci	16.0 KiB	-
attendance	Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_general_ci	32.0 KiB	-
feedetails	Browse Structure Search Insert Empty Drop	2	InnoDB	utf8mb4_general_ci	32.0 KiB	-
paymentdetails	Browse Structure Search Insert Empty Drop	3	InnoDB	utf8mb4_general_ci	32.0 KiB	-
student1	Browse Structure Search Insert Empty Drop	3	InnoDB	utf8mb4_general_ci	32.0 KiB	-
student_info	Browse Structure Search Insert Empty Drop	3	InnoDB	utf8mb4_general_ci	16.0 KiB	-

Total: 6 tables Sum: 12 InnoDB utf8mb4_general_ci 160.0 KiB 0 B

Data base tables

phpMyAdmin

Server: 127.0.0.1 > Database: software > Table: student1

Recent Favorites

Table structure Relation view

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	Roll	int(100)			No	None			Change Drop More
2	Name	varchar(100)	utf8mb4_general_ci		No	None			Change Drop More
3	Class	bigint(100)			No	None			Change Drop More
4	Email	varchar(100)	utf8mb4_general_ci		No	None			Change Drop More
5	Phone_Number	varchar(100)	utf8mb4_general_ci		No	None			Change Drop More
6	Address	varchar(100)	utf8mb4_general_ci		No	None			Change Drop More
7	Blood_Group	varchar(100)	utf8mb4_general_ci		No	None			Change Drop More
8	Gender	varchar(100)	utf8mb4_general_ci		No	None			Change Drop More
9	DOB	date			No	None			Change Drop More
10	Payment	varchar(100)	utf8mb4_general_ci		No	NOT PAID			Change Drop More

Check all With selected: [Browse](#) [Change](#) [Drop](#) [Primary](#) [Unique](#) [Index](#) [Spatial](#) [Fulltext](#)

Add to central columns Remove from central columns

Structure of student1 table

phpMyAdmin

Server: 127.0.0.1 > Database: software > Table: student_info

Recent Favorites

Table structure Relation view

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	Roll	int(100)			No	None			Change Drop More
2	Class	int(100)			No	None			Change Drop More
3	Password	varchar(100)	utf8mb4_general_ci		No	None			Change Drop More
4	Securecode	varchar(100)	utf8mb4_general_ci		No	None			Change Drop More
5	Name	varchar(100)	utf8mb4_general_ci		No	None			Change Drop More
6	Email	varchar(100)	utf8mb4_general_ci		No	None			Change Drop More

Check all With selected: [Browse](#) [Change](#) [Drop](#) [Primary](#) [Unique](#) [Index](#) [Spatial](#) [Fulltext](#)

Add to central columns Remove from central columns

Structure of student credentials table

The screenshot shows the phpMyAdmin interface for the 'attendance' table in the 'software' database. The left sidebar shows various databases and tables. The main area displays the table structure with the following columns:

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	Roll	int(100)			No	None			Change Drop More
2	Class	bigint(100)			No	None			Change Drop More
3	Date	date			No	None			Change Drop More
4	Name	varchar(100)	utf8mb4_general_ci		No	None			Change Drop More
5	Email	varchar(100)	utf8mb4_general_ci		No	None			Change Drop More
6	Status	varchar(100)	utf8mb4_general_ci		No	None			Change Drop More

Buttons at the bottom include: Check all, With selected: Browse, Change, Drop, Primary, Unique, Index, Spatial, Fulltext, Add to central columns, Remove from central columns.

Attendance table

The screenshot shows the phpMyAdmin interface for the 'admin' table in the 'software' database. The left sidebar shows various databases and tables. The main area displays the table structure with the following columns:

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	Name	varchar(100)	utf8mb4_general_ci		No	None			Change Drop More
2	Email	varchar(100)	utf8mb4_general_ci		No	None			Change Drop More
3	Admin_id	varchar(100)	utf8mb4_general_ci		No	None			Change Drop More
4	Password	varchar(100)	utf8mb4_general_ci		No	None			Change Drop More
5	Securecode	varchar(100)	utf8mb4_general_ci		No	None			Change Drop More
6	Phone_Number	varchar(10)	utf8mb4_general_ci		No	None			Change Drop More
7	Gender	varchar(100)	utf8mb4_general_ci		No	None			Change Drop More
8	Address	varchar(100)	utf8mb4_general_ci		No	None			Change Drop More

Buttons at the bottom include: Check all, With selected: Browse, Change, Drop, Primary, Unique, Index, Spatial, Fulltext, Add to central columns, Remove from central columns.

Structure of admin table

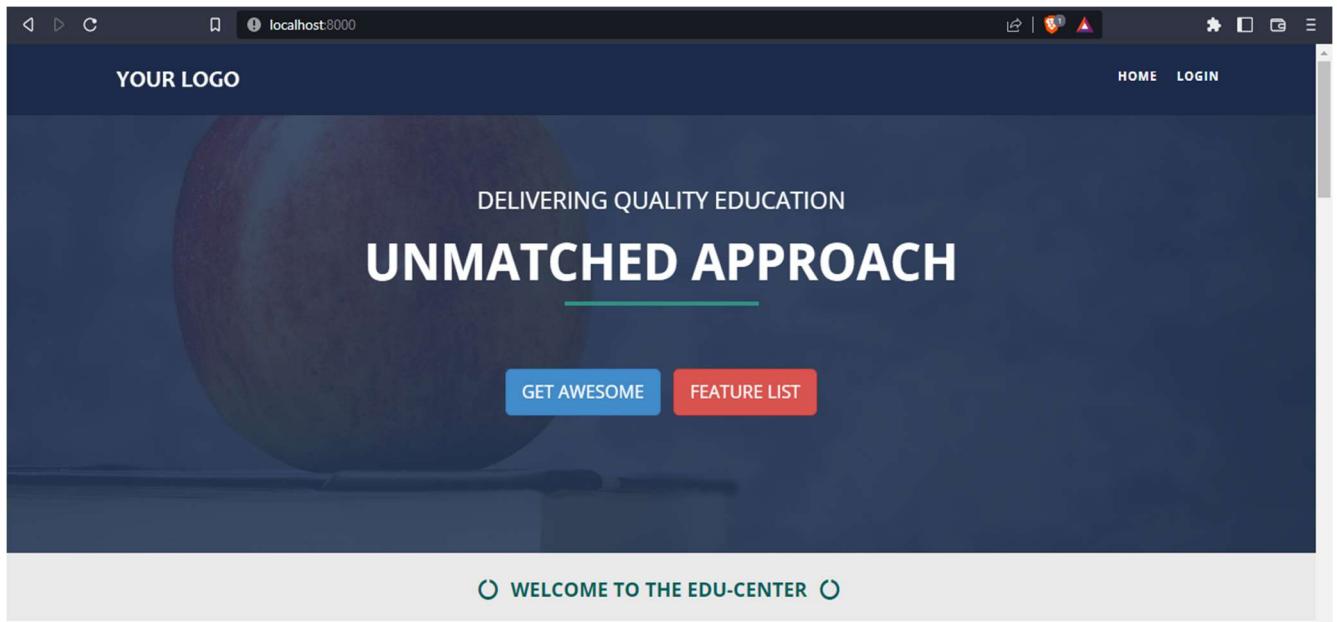
The screenshot shows the phpMyAdmin interface for a MySQL database. The left sidebar lists databases: New, information_schema, mysql, performance_schema, phpmyadmin, software, and several custom databases like student1, student_info, and paymentdetails. The main window is titled 'Table: feedetails' and displays the table structure. The table has two columns: 'Fees' (varchar(100)) and 'Class' (bigint(100)). The 'Structure' tab is selected, showing the columns and their properties. Below the table definition, there are buttons for 'Add' (with a value of 1), 'Print', 'Propose table structure', 'Track table', 'Move columns', 'Normalize', and a dropdown for 'column(s) after Class'. There is also a 'Go' button.

Structure of fee details table

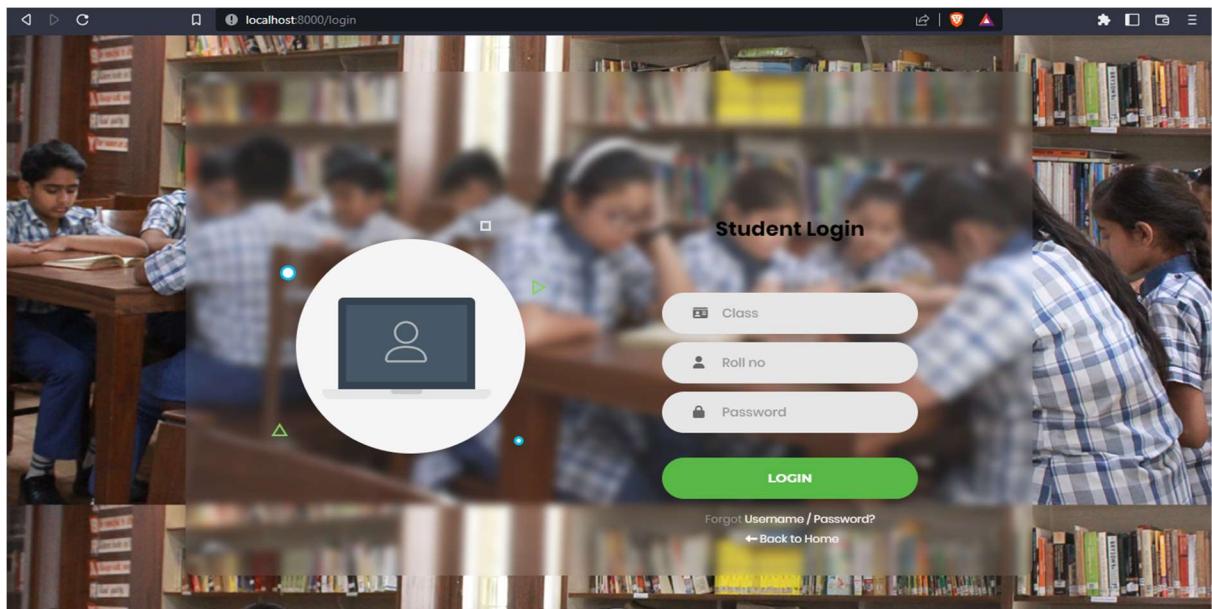
The screenshot shows the phpMyAdmin interface for a MySQL database. The left sidebar lists databases: New, information_schema, mysql, performance_schema, phpmyadmin, software, and several custom databases like student1, student_info, and paymentdetails. The main window is titled 'Table: paymentdetails' and displays the table structure. The table has five columns: 'Transaction_Id' (varchar(100)), 'Roll' (int(100)), 'Class' (bigint(100)), 'Fees' (varchar(100)), and 'Pay_Date' (date). The 'Structure' tab is selected, showing the columns and their properties. Below the table definition, there are buttons for 'Add' (with a value of 1), 'Print', 'Propose table structure', 'Track table', 'Move columns', 'Normalize', and a dropdown for 'column(s) after Pay_Date'. There is also a 'Go' button.

Structure of payment details table

Screenshots of LIVE Mode Execution



Landing page



Login Page (Almost Same for Student and Admin)

The screenshot shows a 'Forgot Password?' page with a large padlock icon at the top. Below it, the title 'Forgot Password?' is displayed, followed by the sub-instruction 'You can reset your password here.' There are five input fields: a text field containing '2054088', a dropdown menu showing '12', a text field with 'aritra' and a red asterisk, and two password fields, both containing '.....'. A blue 'Reset Password' button is centered below the fields. At the bottom, a link 'Back to Login' is visible.

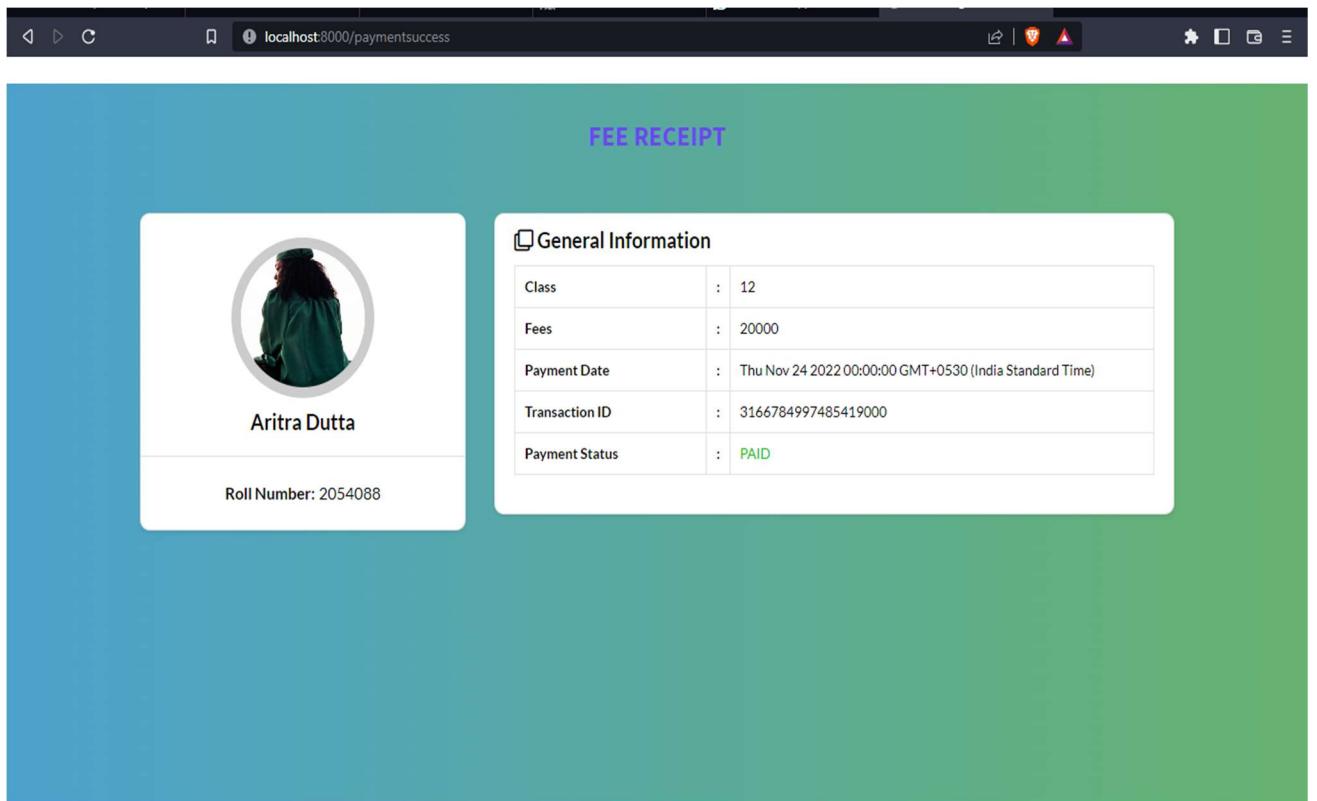
Forgot Password (Almost Same for Student and Admin)

The screenshot shows a 'STUDENT PROFILE PAGE' titled 'YOUR PROFILE'. On the left, there is a circular profile picture of a student named 'Aritra Dutta'. Below the picture, the name 'Aritra Dutta' is displayed, followed by the text 'Roll Number: 2054088'. To the right, a white box contains a table titled 'General Information' with the following data:

General Information	
Class	: 12
Email	: aritra.dutta.it24@heritageit.edu.in
Phone Number	: 9875380964
Address	: 85/2 Rajendra Avenue Uttarpara Hooghly
Date of Birth	: 2002-05-11 (YY/MM/DDDD)
Gender	: Male
Blood	: AB+

At the bottom of the page are two buttons: 'Update' and 'Logout'.

Profile Page (Almost Same for Student and Admin)



FEE RECEIPT PAGE

The screenshot shows an "ADD STUDENT BIO" form with a purple header and white body. The form fields are as follows:

Name	Aritra Dutta
Roll	11
Class	12
Email	aritra@gmail.com
Phone Number	09875380964
Address	85/2 Rajendra Avenue Uttarpara Hooghly

Add Student Page

A screenshot of a web browser window titled "localhost:8000/updatestudent". The main title of the form is "STUDENT DETAILS UPDATE FORM". The form contains the following fields:

- Name: Aritra Dutta
- Roll: 2054088
- Class: 12
- Email: aritra.dutta.it24@heritageit.edu.in
- Phone Number: 9875380964
- Address: 85/2 Rajendra Avenue Uttarpura Hooghly

A dropdown menu is open over the "Address" field, showing the selected address: "Aritra Dutta 85/2 Rajendra Avenue Uttarpura H...". There is also a "Manage addresses..." option in the dropdown.

Update Student Page

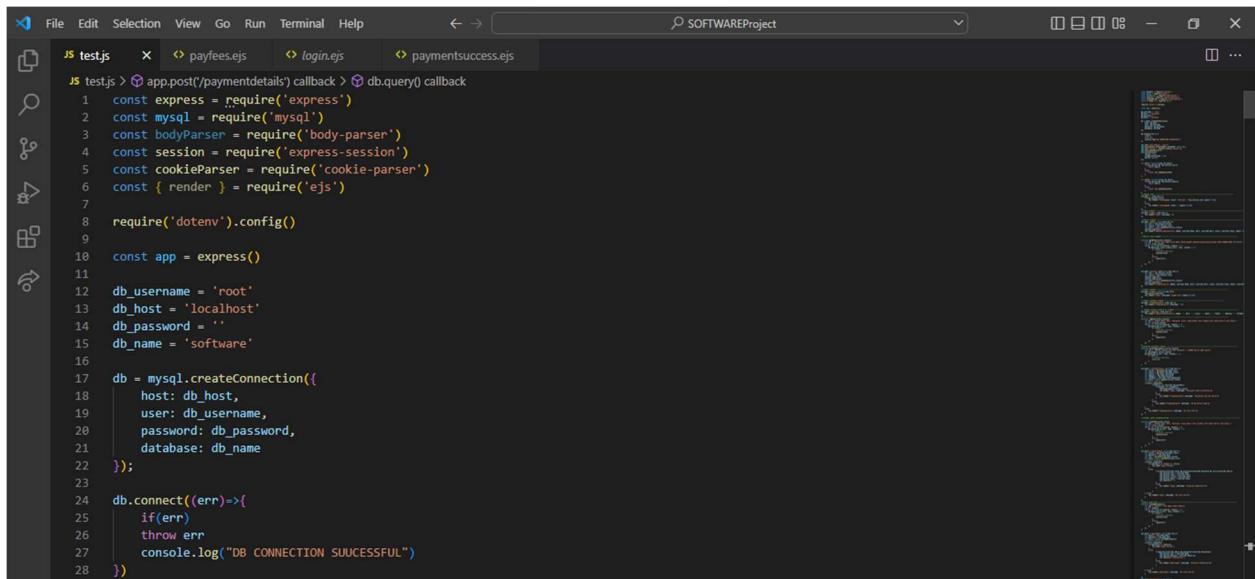
A screenshot of a web browser window titled "localhost:8000/payfees". The form displays the same student information as the previous screenshot, plus a "Fees" field:

- Name: Aritra Dutta
- Roll: 2054088
- Class: 12
- Email: aritra.dutta.it24@heritageit.edu.in
- Phone Number: 9875380964
- Fees: 20000

A red "CONTINUE" button is located at the bottom of the form.

Payment Fees Page

Screenshot of important code segments



```
File Edit Selection View Go Run Terminal Help
JS test.js x payfees.ejs login.ejs paymentsuccess.ejs
JS test.js > app.post('/paymentdetails') callback > db.query() callback
1 const express = require('express')
2 const mysql = require('mysql')
3 const bodyParser = require('body-parser')
4 const session = require('express-session')
5 const cookieParser = require('cookie-parser')
6 const { render } = require('ejs')
7
8 require('dotenv').config()
9
10 const app = express()
11
12 db_username = 'root'
13 db_host = 'localhost'
14 db_password = ''
15 db_name = 'software'
16
17 db = mysql.createConnection({
18   host: db_host,
19   user: db_username,
20   password: db_password,
21   database: db_name
22 });
23
24 db.connect((err)=>{
25   if(err)
26     throw err
27   console.log("DB CONNECTION SUCESSFUL")
28 })
```

Required dependencies & database connection

The image shows two side-by-side screenshots of the Visual Studio Code (VS Code) interface. Both screenshots display the same code editor with dark theme styling.

Top Screenshot (test.js):

```
JS test.js  x  payfees.ejs  login.ejs  paymentsuccess.ejs
JS test.js > app.post('/paymentdetails') callback > db.query() callback
1// ...
178
179 //STUDENT LOGIN AUTHENTICATION -----
180
181 function userData(class1,roll){
182     let sql = `SELECT Name, Roll, Password, Class,Email from student_info where Roll=? and Class=?`;
183     var p = [roll,class1];
184     return new Promise((parameter) => {
185         db.query(sql,p,(err, rows, fields) => {
186             if (!err) {
187                 //console.log(rows)
188                 resolve(rows);
189             }
190             else {
191                 reject(err);
192             }
193         })
194     })
195 }
196
197 app.post('/loginstudent',async(req,res)=>{
198     let class1 = parseInt(req.body.class);
199     let password = req.body.pass;
200     let roll = parseInt(req.body.rollno);
201     let users = await userData(class1,roll);
202     //console.log(users);
203     if(users.length>0){
204         if(!password || !class1 || !roll){
205             res.send('Login Failed');
206         }
207         else{
208             if(class1==users[0].Class && password==users[0].Password && roll==users[0].Roll){
209                 req.session.user = users[0].Name;
210                 req.session.class = users[0].Class;
211                 req.session.rollno = users[0].Roll;
212                 req.session.email = users[0].Email;
213                 res.redirect('/');
214             }
215             else{
216                 res.render('login',{message: "Invalid Credentials"});
217             }
218         }
219     }
220     else{
221         res.render('login',{message: "No such user"});
222     }
223 }
224 })
```

Bottom Screenshot (login.ejs):

```
JS test.js  x  payfees.ejs  login.ejs  paymentsuccess.ejs
JS test.js > app.post('/paymentdetails') callback > db.query() callback
209     req.session.user = users[0].Name;
210     req.session.class = users[0].Class;
211     req.session.rollno = users[0].Roll;
212     req.session.email = users[0].Email;
213     res.redirect('/');
214 }
215 else{
216     res.render('login',{message: "Invalid Credentials"});
217 }
218
219 } else{
220     res.render('login',{message: "No such user"});
221 }
222
223 }
```

Student login(almost same for admin)

```
JS test.js > app.post('/paymentdetails') callback > db.query() callback
283 app.get('/forgotpasswordadmin',(req,res)->{
284   res.render('forgotpasswordadmin',{message: ''})
285 })
286
287 function resetting1(newpass,email1){
288   let sql = `UPDATE admin SET Password = ? WHERE Email=?`
289   let p=[newpass,email1]
290   db.query(sql,p,(err, rows, fields) => {
291     if (err) {
292       //console.log(rows)
293       throw err
294     }
295   })
296 }
297
298 app.post('/resetpasswordadmin',async(req,res)->{
299   let email1 = req.body.email
300   let secure = req.body.securecode
301   let newpass = req.body.password
302   let cnewpass = req.body.confirmpassword
303   let users = await fpdata1(email1)
304   //console.log(users)
305   if(users.length>0){
306     if(secure === users[0].Securecode){
307       if(newpass === cnewpass){
308         resetting1(newpass,email1)
309         res.render('adminlogin',{message: 'Password reset Successful'})
310       } else{
311         res.render('forgotpasswordadmin',{message: 'Passwords did not match'})
312       }
313     } else{
314       res.render('forgotpasswordadmin',{message: 'Wrong Secure Code'})
```

```
JS test.js > fpdata1
314   }
315   else{
316     res.render('forgotpasswordadmin',{message: 'Wrong Secure Code'})
317   }
318 }
319 else{
320   res.render('forgotpasswordadmin',{message: 'No such user'})
321 }
322 }

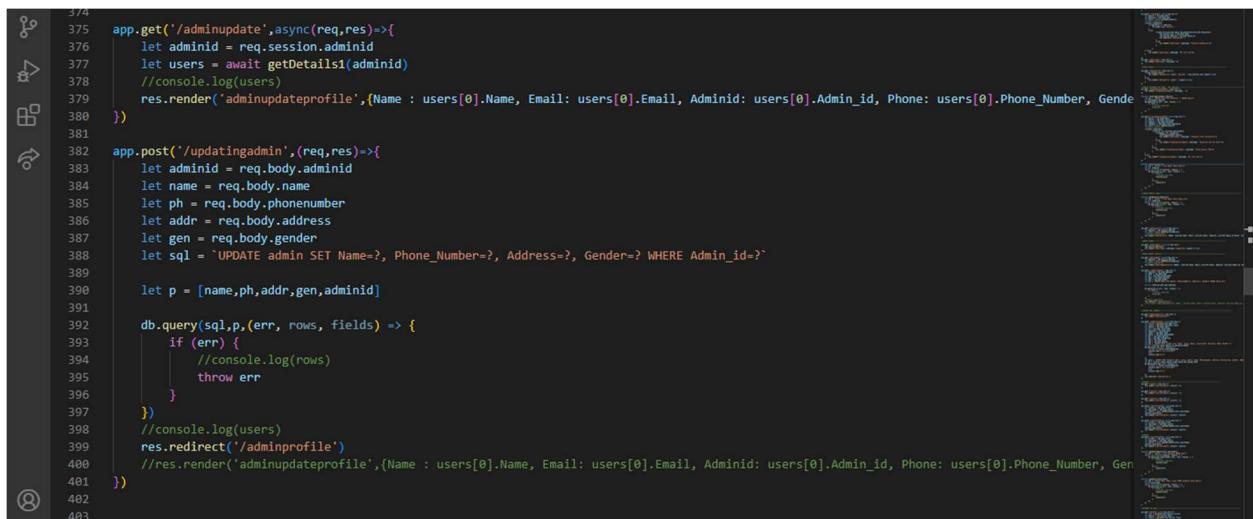
323 function fpdata1(email1){
324   let sql = `SELECT * from admin where Email=?`
325   var p = [email1]
326   return new Promise((resolve, reject) => {
327     db.query(sql,p,(err, rows, fields) => {
328       if (!err) {
329         //console.log(rows)
330         resolve(rows)
331       } else {
332         reject(err)
333       }
334     })
335   })
336 }
337 }
338 }
```

Forgot password(almost same for student and admin)



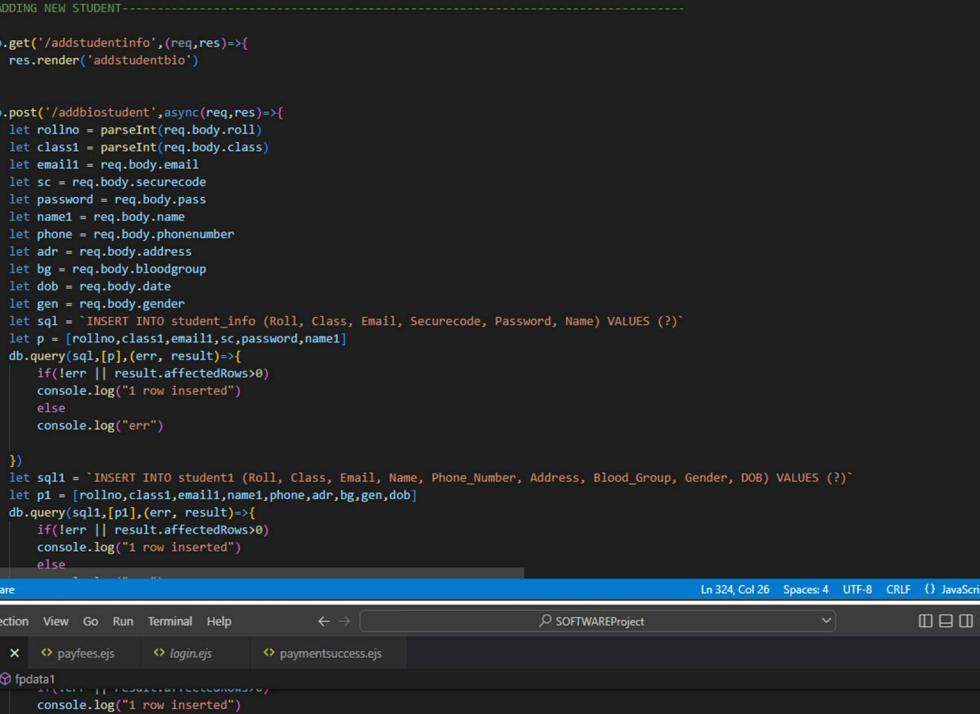
```
341     function getDetails1(adminid){
342       let sql = `SELECT * from admin where Admin_id=?`;
343       var p = [adminid];
344       return new Promise((resolve, reject) => {
345         db.query(sql,p,(err, rows, fields) => {
346           if (!err) {
347             //console.log(rows)
348             resolve(rows);
349           } else {
350             reject(err);
351           }
352         })
353       })
354     }
355   }
356 }
357
358 app.get('/adminprofile',async(req,res)=>{
359   let adminid = req.session.adminid
360   let users = await getDetails1(adminid)
361   //console.log(adminid)
362   res.render('adminprofile',{Name: users[0].Name, Email: users[0].Email, Adminid: users[0].Admin_id,Phone: users[0].Phone_Number, Address: user
363 })
364 })
365
```

Profile page(almost same for student and admin)



```
374
375 app.get('/adminupdate',async(req,res)=>{
376   let adminid = req.session.adminid
377   let users = await getDetails1(adminid)
378   //console.log(users)
379   res.render('adminupdateprofile',{Name : users[0].Name, Email: users[0].Email, Adminid: users[0].Admin_id, Phone: users[0].Phone_Number, Gender: users[0].Gender})
380 })
381
382 app.post('/updatingadmin',(req,res)=>{
383   let adminid = req.body.adminid
384   let name = req.body.name
385   let ph = req.body.phonenumber
386   let addr = req.body.address
387   let gen = req.body.gender
388   let sql = `UPDATE admin SET Name=?, Phone_Number=?, Address=?, Gender=? WHERE Admin_id=?`;
389
390   let p = [name,ph,addr,gen,adminid]
391
392   db.query(sql,p,(err, rows, fields) => {
393     if (err) {
394       //console.log(rows)
395       throw err;
396     }
397   })
398   //console.log(users)
399   res.redirect('/adminprofile')
400   //res.render('adminupdateprofile',{Name : users[0].Name, Email: users[0].Email, Adminid: users[0].Admin_id, Phone: users[0].Phone_Number, Gen
401 })
402
403
```

Updating details(almost same for student and admin)



The screenshot shows a Microsoft Edge browser window with the following details:

- Title Bar:** SOFTWAREProject
- Address Bar:** testjs
- Content Area:** A 404 error page titled "Page Not Found". The page includes the URL "http://127.0.0.1:5000/testjs", a search bar, and a link to "View Source".
- Bottom Status Bar:** Ln 324, Col 26 | Spaces: 4 | UTF-8 | CRLF | () JavaScript | Go Live | F11

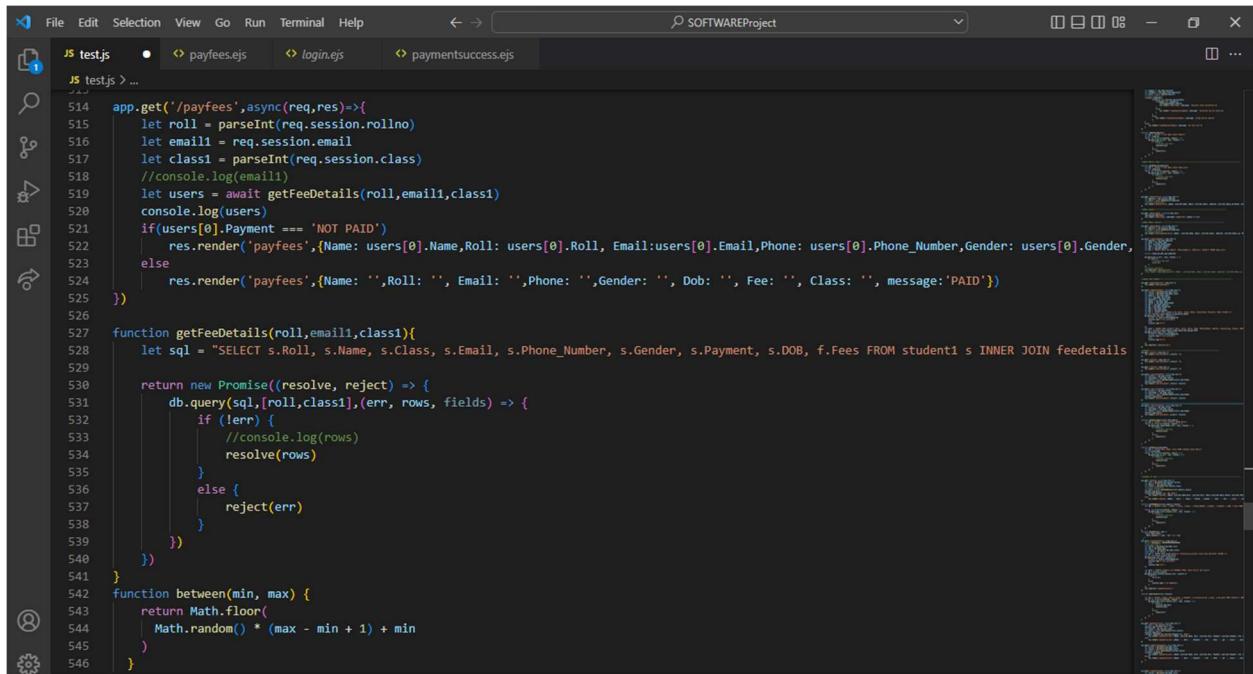
The screenshot also captures the developer tools interface on the right side of the browser, showing the DOM tree, Element inspector, and Network tab.

Adding students

```
473 app.post('/searchstudent1',async(req,res)=>{
474   let filter1 = req.body.filter
475   let searched1 = req.body.search
476   let result = await getSearcheds(filter1,searched1)
477   console.log(result)
478   res.render('searchstudent1',{result: result})
479 })
480
481 function getSearcheds(filter1,searched1){
482   let sql = 'SELECT * from student1 WHERE Roll=?'
483   return new Promise((resolve, reject) => {
484     db.query(sql,[searched1],(err, rows, fields) => {
485       if (!err) {
486         //console.log(rows)
487         resolve(rows)
488       }
489       else {
490         reject(err)
491       }
492     })
493   })
494 }
```

Searching students

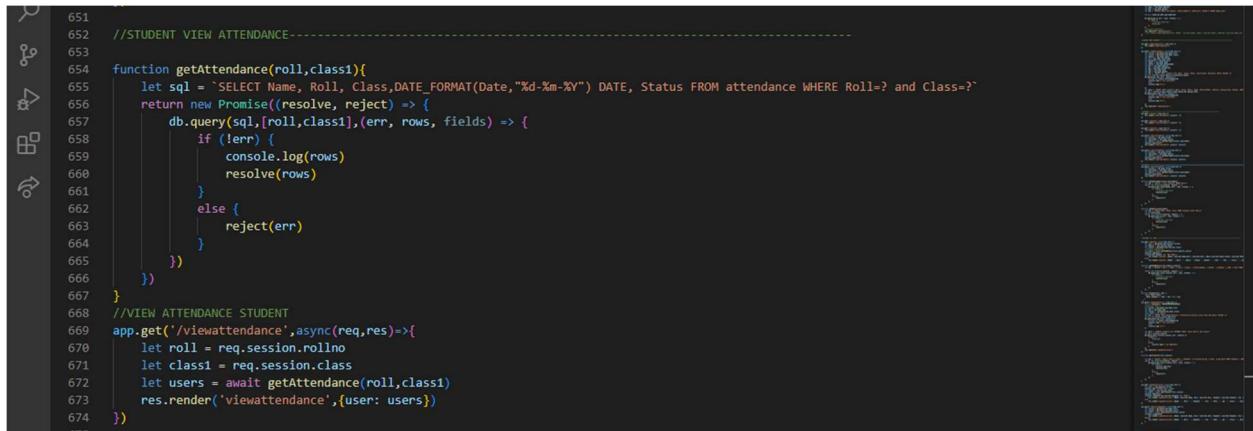
```
JS test.js  ● payfees.ejs  ◊ login.ejs  ◊ paymentsuccess.ejs
JS test.js > ...
545   }
546 }
547 app.post('/paymentdetails',(req,res)=>{
548   let t = between(1, 9999999999999999)
549   //req.user.tid= t
550   let rollno = parseInt(req.body.roll)
551   let fee = req.body.Fees
552   let class1 = parseInt(req.body.class)
553   const date = new Date()
554   let sql = `INSERT INTO paymentdetails (Transaction_Id,Roll,Class,Fees,Pay_Date) VALUES (?)`
555   let p = [t,rollno,class1,fee,date]
556   db.query(sql,[p],(err, result)=>{
557     if(!err || result.affectedRows>0)
558       console.log("1 row inserted")
559     else
560       console.log("err")
561   })
562
563   let sql1 = 'UPDATE student1 set PAYMENT='PAID' where Roll=? and Class=?'
564   let p1 = [rollno,class1]
565   db.query(sql1,[rollno,class1],(err, result)=>{
566     if(err) {
567       throw err
568     }
569     else{
570       console.log("1 row Updated")
571     }
572   })
573   res.redirect('/paymentsuccess')
574 })
575
576 function gettransact(roll,class1){
577
578   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
579
580   function gettransact(roll,class1){
581
582     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
583     return new Promise((resolve, reject) => {
584       db.query(sql,[roll,class1],(err, rows, fields) => {
585         if (!err) {
586           console.log(rows)
587           resolve(rows)
588         }
589         else {
590           reject(err)
591         }
592     })
593     app.get('/paymentsuccess',async(req,res)=>{
594       let roll = req.session.rollno
595       console.log('PAYMENT ROLL',roll)
596       let class1 = req.session.class
597       let users = await gettransact(roll,class1)
598       console.log(users)
599       if(users.length>0 && users[0].Payment === 'PAID')
600         res.render('paymentsuccess',{Name: users[0].Name, Roll: users[0].Roll, Payment: users[0].Payment, Tid: users[0].Transaction_Id, Fees: user
601       else
602         res.render('paymentsuccess',{Name: '', Roll: '', Payment: '', Tid: '', Fees: '', pd: '', Class: '', message:'NOT YET PAID'})
```



```
JS test.js > ...
514 app.get('/payfees',async(req,res)=>{
515   let roll = parseInt(req.session.rollno)
516   let email1 = req.session.email
517   let class1 = parseInt(req.session.class)
518   //console.log(email1)
519   let users = await getFeeDetails(roll,email1,class1)
520   console.log(users)
521   if(users[0].Payment === 'NOT PAID')
522     res.render('payfees',{Name: users[0].Name,Roll: users[0].Roll, Email:users[0].Email,Phone: users[0].Phone_Number,Gender: users[0].Gender,
523     else
524       res.render('payfees',{Name: '',Roll: '', Email: '',Phone: '',Gender: '', Dob: '', Fee: '', Class: '', message:'PAID'})
525   })
526
527   function getFeeDetails(roll,email1,class1){
528     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
529
530     return new Promise((resolve, reject) => {
531       db.query(sql,[roll,class1],(err, rows, fields) => {
532         if (!err) {
533           //console.log(rows)
534           resolve(rows)
535         }
536         else {
537           reject(err)
538         }
539       })
540     })
541     function between(min, max) {
542       return Math.floor(
543         Math.random() * (max - min + 1) + min
544       )
545     }
546   }

```

Fees payment



```
651 //STUDENT VIEW ATTENDANCE
652
653 function getAttendance(roll,class1){
654   let sql = `SELECT Name, Roll, Class,DATE_FORMAT(Date,"%d-%m-%Y") DATE, Status FROM attendance WHERE Roll=? and Class=?`
655   return new Promise((resolve, reject) => {
656     db.query(sql,[roll,class1],(err, rows, fields) => {
657       if (!err) {
658         console.log(rows)
659         resolve(rows)
660       }
661       else {
662         reject(err)
663       }
664     })
665   })
666 }
667 //VIEW ATTENDANCE STUDENT
668 app.get('/viewattendance',async(req,res)=>{
669   let roll = req.session.rollno
670   let class1 = req.session.class
671   let users = await getAttendance(roll,class1)
672   res.render('viewattendance',{user: users})
673 })
674
675
```

Student view attendance

BIBLIOGRAPHY

BOOKS/REFFERENCES	AUTHORS/SOURCE
<ol style="list-style-type: none">1. Software engineering2. INTERNET<ul style="list-style-type: none">○ COLORIB○ W3C○ Free CSS○ Font Awesome	<p>Sajan Mathew</p> <p>https://colorlib.com/</p> <p>http://www.w3.org/</p> <p>https://www.freecss.com/free-css-templates</p> <p>https://fontawesome.com/</p>

