Department of Computer Engineering

CMSE353 - Security of Software Systems

SUPERVISOR: Prof. Dr. Alexander Chefranov

Term Project

TEAM MEMBERS:

Hassan El Abdallah - 18700656 (TEAM LEADER)

Abdallah Waleed Mustafa Khasseb - 19700434

Abdulaziz Ahmed Hussein Binafif - 19701169

Mohammed Hossein Bokaye - 19701190

Table of contents

1. Problem definition

1.1 Problem

Universities often have large enrollments and it can be difficult for instructors to accurately track student attendance in their classes. This can lead to issues with course completion and academic performance, as well as difficulties in tracking absences for administrative or compliance purposes.

1.2 Objective

The objective of a university/emu attendance management system is to provide a reliable and efficient way for instructors to track student attendance in their classes. The system should allow instructors to easily record attendance for each class session, as well as view attendance data for individual students or for the class as a whole. The system should also provide tools for analyzing students and courses data, encrypting and decrypting data.

2. Task distribution

2.1 Description

Team work is an important aspect of any successful project, and the development of a university/emu attendance management system is no exception. In this project, team members will work together to plan, design, and implement the system, as well as writing tests.
To facilitate team work, it may be helpful to establish clear roles and responsibilities for each team member, as well as establish regular meetings and check-ins to keep everyone informed about the project's progress. In addition, the team should use a project management tool or software to track tasks and deadlines, and to share updates and collaborate on tasks.

2.2 Meetings and timeline

Date of meeting	Discussion
18/12/2022	This meeting held at the beginning of the project to introduce the team, review the project goals and objectives, and discuss the project plan.
20/12/2022	This meeting is to review the project plan, discuss progress and any issues that have arisen, and make any necessary adjustments to the plan.
23/12/2022	Review and discuss the design of the system, including the user interface, functionalities, and overall user experience.
26/12/2022	Discuss the progress of the development work and any issues that have arisen.
27/12/2022	This meeting is to review the progress of the development and discuss the next functionalities
28/12/2022	Reviewing backend code and fixing errors
29/12/2022	This meeting held to review and discuss the results of testing and to make any necessary adjustments to the system.
30/12/2022	This meeting held to discuss building frontend and user interface of the system
01/01/2023	Reviewing backend and frontend, and documenting the project

2.3 Work distribution

- **Hassan El Abdallah** was responsible for distributing the work, designing the database structure, functional requirements, and UML diagrams of the system.
- **Mohammed Hossien Bokaie** was responsible for developing the backend, handling encryption/decryption algorithms, and designing the user interface of the system.
- **Abdulaziz Ahmed Hussien** was responsible writing tests for the system and contributing in the frontend development.

- **Abdallah Waleed Mustafa** was responsible of doing research, and documenting tests results and the progress of the team members.

3. Description of algorithms

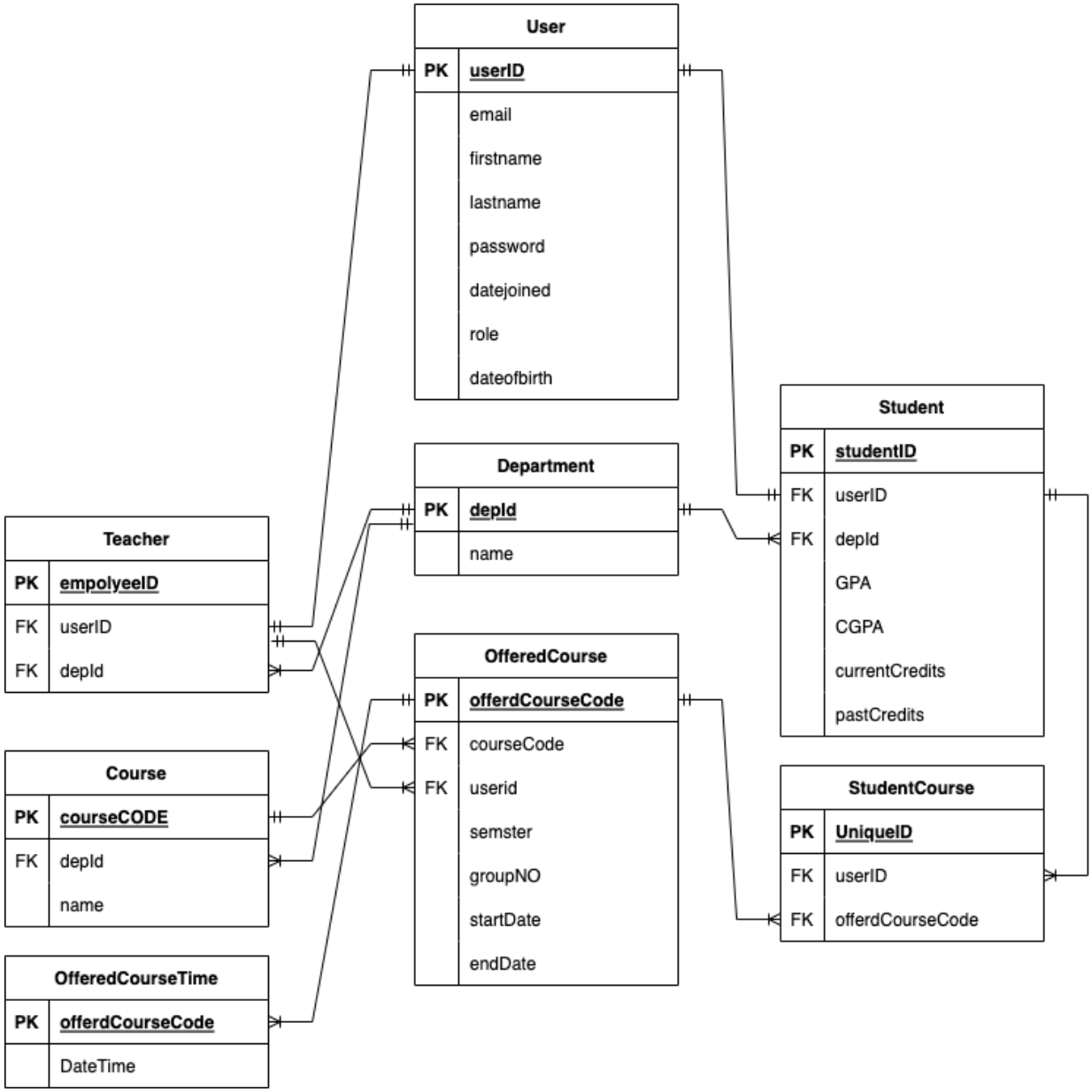
DES (Data Encryption Standard)

is a symmetric-key block cipher algorithm that was developed in the 1970s and is now considered to be outdated. It uses a 56-bit key to encrypt and decrypt data, and operates on fixed-size blocks of data (64 bits).

The DES algorithm works by applying a series of transformations, or "rounds," to the input data using the key. Each round consists of several steps, including permutation, substitution, and XOR operations. The number of rounds and the specific operations used depend on the key being used.

We basically used this algorithm to encrypt all data when stored in the database, decrypt data when fetched from the database to display it to the user.

4. Description of database structure



5. Description of tools used

- **VSCode**: VSCode (Visual Studio Code) is a free, open-source code editor developed by Microsoft. It is available for Windows, macOS, and Linux, and is widely used by developers for writing and debugging code. VSCode supports a variety of programming languages and includes features such as code completion, syntax highlighting, and integrated debugging.

- **macOS**: macOS is the operating system that runs on Apple's Mac computers. It is a Unix-based operating system that is known for its user-friendly interface and wide range of built-in tools and features.

- **MySQL**: MySQL is a popular open-source database management system. It is used to store, organize, and retrieve data, and is commonly used in web applications. MySQL supports a variety of programming languages and can be used in a wide range of applications.

- **Node.js**: Node.js is a JavaScript runtime that allows developers to run JavaScript on the server side. It is designed to be lightweight and efficient, and is commonly used for building web applications.

- **Express.js**: Express.js is a web application framework for Node.js. It is designed to make it easier to develop web applications by providing a set of tools and utilities for handling common tasks such as routing, middleware, and error handling.

- **GitHub**: GitHub is a web-based platform for version control and collaboration. It allows developers to store and manage their code repositories, as well as track changes and collaborate with other developers. GitHub is widely used by developers for sharing and collaborating on code projects.

6. Description of the program

Before we start, link of the project repository (backend) on Github: https://github.com/hsnkh12/emu-attendance-management-system

DES encryption/decryption class

This class is responsible to encrypt and decrypt data using DES algorithm provided by NodeJs “crypto” library. It has 2 methods, one for data encryption, and the other for data decryption, using provided key.

```
const key = Buffer.from("d0e276d0144890d3", "hex");

class Des {
    static async encrypt(text) {
        if (text != null) {
            const cipher = crypto.createCipheriv(algorithm, key, null);
            let encrypted = cipher.update(text, "utf8", "hex");
            encrypted += cipher.final("hex");
            return encrypted;
        }
        return null;
    }
    static async dencrypt(encrypted) {
        try {
            if (encrypted != null) {
                const decipher = crypto.createDecipheriv(algorithm, key, null);
                let decrypted = decipher.update(encrypted, "hex", "utf8");
                decrypted += decipher.final("utf8");
                return decrypted;
            }
            return encrypted;
        } catch {
            return encrypted;
        }
    }
}
```

Password manager class

This class is responsible for hashing passwords using “bycrypt” library provided by nodeJs. It has to methods, one for hashing password, the other is for comparing hashed password.

```
6   class PasswordManager {
7
8       static async hashPassword(password) {
9
10          const salt = await bcrypt.genSalt(6);
11          const hashed = await bcrypt.hash(password, salt);
12
13          return hashed
14      }
15
16      static async comparePassword(password, hashedPassword) {
17          return await bcrypt.compare(password, hashedPassword);
18      }
19
20
21  }
```

Login controller function

This function/controller is responsible for authenticating the user for login by checking provided credentials, by checking the username and comparing the password. If given information is valid, a new JWT token will be generated and will be returned as a response for the user to use it in other requests.

```
const loginController = async(req, res) => {

    let body = req.body;
    const username = await Des.encrypt(body.username);
    const password = body.password;

    try {

        // finding user by username provided in body
        let user = await getUserByUserId(username);
        if(user == null){
            return res.status(403).send({ Message: "User with this username not found" });
        }

        // Compare passwords
        const passwordIsValid = await PasswordManager.comparePassword(
            password,
            await Des.dencrypt(user.password)
        );

        // Check the validation of the password
        if (passwordIsValid) {

            // Generate a jwt token for the user using his data
            const user_to_enc = { userId: user.userId, role: user.role };
            jwt.sign(user_to_enc,
                JWT_SECRET_KEY, { expiresIn: "30m" },
                (err, token) => {
                    return res.json({
                        token,
                    });
                }
            );

        } else {
            return res.status(401).send({ Message: "Password incorrect" });
        }

    } catch (error) {
        console.log(error);
        return res.status(401).send({ Message: "Invalid credentials" });
```

Create student controller function

This function will be responsible for adding new student to the database, and admin is only authorized to add a new student as we can see in line 95. **Student** will be created by first checking if the username is taken, then encrypt student data, and save it to the database.

```
88  const createStudentController = async(req, res) => {
89
90      const body = req.body;
91
92      try {
93
94          // Check if role not Admin
95          if (req.role != "A") {
96              return res
97                  .status(403)
98                  .send({ Message: "Only admin can add new student" });
99          }
100
101         // encrypt username/userId and check it it's taken by other user
102         const userId = await Des.encrypt(body.userId);
103         let usernameIsTaken = await checkUsernameIsTaken(userId);
104
105         if (usernameIsTaken) {
106             return res.status(403).send({ Message: "This username/userId already exists" });
107         }
108
109         // Create new student, decrypt, and save
110         const student = await createNewEncryptedStudent(body);
111         await student.save();
112         return res.status(200).send({ Message: "New student added" });
113
114     } catch (error) {
115         console.log(error);
116         return res.status(503).send({ Message: "Something went wrong" });
117     }
118 };
```

```
96  const createNewEncryptedStudent = async (body) => {
97
98      return await Student.create({
99          firstName: await Des.encrypt(body.firstName),
100         lastName: await Des.encrypt(body.lastName),
101         studentId: await Des.encrypt(body.studentId),
102         email: await Des.encrypt(body.email),
103         password: await Des.encrypt(
104             await PasswordManager.hashPassword(body.password)
105         ),
106         role: await Des.encrypt("S"),
107         dateJoined: await Des.encrypt(body.dateJoined.toString()),
108         userId: await Des.encrypt(body.userId),
109         depId: await Des.encrypt(body.depId),
110     });
111 }
```

Create teacher controller function

This function will be responsible for adding new teacher in the database, and it's also authorized only by **admin**. Check the username if it's taken, then encrypt the data and save it in the database.

```
const createTeacherController = async(req, res) => {

    const body = req.body;

    try {

        // Check the role of the user if it's not Admin
        if (req.role != "A") {
            return res
                .status(403)
                .send({ Message: "Only Admin is authorized to add new teacher" });
        }

        // Get username added in body and check of it's taken by other user
        const userId = await Des.encrypt(body.userId);
        let usernameIsTaken = await checkUsernameIsTaken(userId);

        // Check if it's already taken
        if (usernameIsTaken) {
            return res.status(403).send({ Message: "someone took this username" });
        }

        // Check if role added in body if its either a teacher or chair
        if (body.Urole == "T" || body.Urole == "C") {

            // Create, encrypt, and save the new teacher
            const teacher = await createNewEncryptedTeacher(body,userId);
            await teacher.save();

            return res.status(200).send({ Message: "New Teacher added" });

        } else {
            return res.json({ Message: "role should be T or C" });
        }

    } catch (error) {
        console.log(error);
        return res.status(404).send({ Message: "Something went wrong" });
    }
};
```

```
const createNewEncryptedTeacher = async (body, userId) =>{

    return await Teacher.create({
        firstName: await Des.encrypt(body.firstName),
        lastName: await Des.encrypt(body.lastName),
        employeeId: await Des.encrypt(body.employeeId),
        email: await Des.encrypt(body.email),
        password: await Des.encrypt(
            await PasswordManager.hashPassword(body.password)
        ),
        role: await Des.encrypt(body.Urole),
        dateJoined: await Des.encrypt(body.dateJoined.toString()),
        userId: userId,
        depId: await Des.encrypt(body.depId),
    });

}
```

Course enroll function controller

This function will be responsible for enrolling a student into a specific course. First role is checked, and it's only authorized by **student** and **admin** as shown in line 331. If it was student, it will check if enrolling for himself, otherwise if admin, validate the information and enroll the student.

```
326  const enrollCourseController = async(req, res) => {
327
328      try {
329
330          // Check if the role neither student nor admin
331          if (req.role != "S" && req.role != "A"){
332
333              return res.status(403).send({
334                  Message: "Only Admin and student authorized to enroll for a course",
335              });
336          }
337
338          let studentUserId = null;
339
340          // Check if role is Student
341          if (req.role == "S") {
342
343              const student = await getStudentByUserId(req.userID);
344
345              // Check if user is enrolling for himself
346              if (student.userId != req.userId) {
347                  return res.status(403).send({
348                      Message: "you can only enroll for yourself",
349                  });
350              }
351
352              // Assign requested user as student to be enrolled
353              studentUserId = req.userId;
354
355          } else {
356
357              // Encrypt student id and get student by the id
358              const studentId = await Des.encrypt(req.body.studentId);
359              const student = await getStudentByStdId(studentId);
360
361              // Assign fetched student as student to be enrolled
362              studentUserId = student.userId;
363          }
```

Continue

```
364
365         // Create new student course with assigned student user id
366         const enrolledCourse = await StudentCourse.create({
367             offeredCourseCode: await Des.encrypt(req.params.offeredCourse),
368             userId: studentUserId,
369         });
370         await enrolledCourse.save();
371         return res.status(200).send({ Message: "Student enrolled" });
372
373
374     } catch (error) {
375
376         if (error.name == "SequelizeUniqueConstraintError") {
377             return res.status(403).send({ Message: "student is already enrolled" });
378         }
379         console.log(error.name);
380         return res.status(404).send({ Message: "Something went wrong" });
381     }
382 };
```

Get list of student function controller

This function will be responsible for returning back list of students based on the role of the user. If the role is teacher, it will return all students within the same offered course. If role is **Chair**, it will return all students within the same department, otherwise if **admin**, it will return all students.

```
21  const listStudentsController = async(req, res) => {
22
23      try {
24
25          // Check if the role is Teacher, Chair, or Admin
26          if (req.role == "T") {
27
28              // encrypt offered course id provided in body
29              const offeredCourseID = await Des.encrypt(req.body.offered_course);
30
31              // Check if it's not provided, send a message
32              if (!offeredCourseID) {
33                  return res
34                      .status(400)
35                      .send({ Message: "Offered course id must be provided in the URL" });
36              }
37
38              // Get offered course, and get teacher that teaches this offered course
39              const offeredCourse = await getOfferedCourseById(offeredCourseID);
40              const teacher = await getTeacherByUserId(offeredCourse.userId);
41
42              // Check if the user is NOT the teacher
43              if (teacher.userId != req.userID) {
44                  return res.status(403).send({
45                      Message: "Only teachers associated with this course can view its students",
46                  });
47              }
48
49              const studentCoursesId = await getStudentCourseByOffCourseId(offeredCourse.offeredCourseCode);
50
51              // Get students associated with this course, decrypt them, and send as a response
52              const students = await getAllStudentsById(studentCoursesId.studentId);
53              const decryptedStudents = await getDecryptedStudents(students);
54              return res.json(decryptedStudents);
55
56
57          } else if (req.role == "C") {
```

Continue

```
57          } else if (req.role == "C") {
58
59
60              const teacher = await getTeacherByUserId(req.userID);
61
62              // Get students with the department as the user/chair, decrypt, and send
63              const students = await getAllStudentsByDepId(teacher.depId);
64              const decryptedStudents = await getDecryptedStudents(students);
65              return res.json(decryptedStudents);
66
67          } else if (req.role == "A") {
68
69              // Get all students, decrypt, and return them
70              const students = await Student.findAll();
71              const decryptedStudents = await getDecryptedStudents(students);
72              return res.json(decryptedStudents);
73
74          } else {
75              return res.status(403).send({
76                  Message: "Only teachers, chairs, and admin can view students",
77              });
78          }
79
80      } catch (error) {
81          console.log(error);
82          return res.status(503).send({ Message: "Something went wrong" });
83      }
84  };
```

Get list of attendance function controller

This function will be responsible for getting attendance list of students. Attendance returned is for a specific offered course, and it depends on the role of the user. If **teacher**, it will return list of attendance for a course he/she teaches. If **chair**, it will return all students attendance within the same department. If **parent**, it will return all child student attendance. Otherwise if **admin**, it will return them all.

```
13  const listAttendanceController = async(req, res) => {
14
15      const offeredCourseCode = await Des.encrypt(req.params.offeredCourseID)
16
17      // Check if offered course id and date mentioned in the url
18      if (!offeredCourseCode) {
19          return res.status(400).send({ 'Message': 'Offered course id should be provided in the URL' })
20      }
21
22      try {
23
24          // Get the attendances related to offered course
25          const attendanceList = await getAllAttendanceByOfferedCourseCode(offeredCourseCode)
26
27          // Check if role is either Teacher, Chair, Parent, or Admin
28          if (req.role == 'T') {
29
30              // Get offered course related to offered course code provided
31              const offeredCourse = await getOfferedCourseById(offeredCourseCode);
32
33              // Check if teacher does NOT teach this course
34              if (offeredCourse.userId != req.userID) {
35                  return res.status(403).json({ 'Message': 'Only teacher associated with this course can check attendance details' })
36              }
37
38
39          } else if (req.role == 'C') {
40
41              const offeredCourse = await getOfferedCourseById(offeredCourseCode);
42              const course = await getCourseById(offeredCourse.courseCode);
43              const chair = await getTeacherByUserId(req.userID);
44
45              // Check if this course is NOT in the same department as the chair
46              if (chair.depId != course.depId) {
47                  return res.status(403).json({ 'Message': 'this course is not in your department' })
48              }
49
```

Continue

```
50          } else if (req.role == 'P') {
51
52              const offeredCourse = await getOfferedCourseById(offeredCourseCode);
53              const course = await getCourseById(offeredCourse.courseCode);
54              const parent = await getParentByUserId(req.userID);
55
56              // Decrypt the courses
57              const decryptedAttendances = await Promise.all(
58                  attendanceList.map(async(attendance) => {
59                      const student = await getStudentByUserId(attendance.userId);
60
61                      // Return only courses which belong to parent's student child
62                      if (student.userId == parent.studentUserId) {
63                          return {
64                              CourseCode: await Des.dencrypt(course.courseCode),
65                              date: await Des.dencrypt(attendance.date),
66                              isPresent: await Des.dencrypt(attendance.isPresent),
67                              studentId: await Des.dencrypt(student.studentId),
68                          }
69                      }
70                  })
71              );
72              return res.json(decryptedAttendances.filter((element) => element != null));
73
74          } else if (req.role != 'A') {
75              return res.status(403).send({ 'Message': 'Only admin, chairs, and teachers are allowed to check the attendance of the students' })
76          }
77
78          // Decrypt the attendances and return them
79          const decryptedAttendances = await getDecryptedAttendances(attendanceList);
80          return res.json(decryptedAttendances);
81
82      } catch (error) {
83          console.log(error)
84          return res.status(500).send({ 'Message': 'Something went wrong' })
85      }
86
```

Get student information controller

This function is responsible for getting student detail information, I depends on the role of the requested user. If user is **Chair**, he/she I only authorized to get student with the same department information. If role is **parent**, they can only view child student information. If role is **Student**, he/she I only authorized to view their information. Otherwise if **admin**, he is authorized for all.

```
const getStudentDetailController = async(req, res) => {

    const queryParams = req.params;

    try {

        // Check if role is Chair, Parent, Student, or Admin
        if (req.role == "C") {

            const teacher = await getTeacherByUserId(req.userID);

            // encrypt student id provided in URL, and get student with this id
            const studentId = await Des.encrypt(queryParams.studentID);
            const student = await getStudentByStdId(studentId);

            // Check if the student is NOT in the same department as the teacher
            if (student.depId != teacher.depId) {
                return res.status(403).send({
                    Message: "Only chair associated with this department can view the student information",
                });
            }

            // Decrypt student, and return it
            const decryptedStudent = await getDecryptedStudent(student);
            return res.json(decryptedStudent);

        } else if (req.role == "P") {

            // Get parent by requested user's id, and encrypt student id provided in URL
            const parent = await getParentByUserId(req.userId);
            const studentId = await Des.encrypt(queryParams.studentID);

            // Check if students's parent is NOT the user
            if (studentId != parent.studentId) {

                return res.status(403).send({
                    Message: "You can only check info of your student",
                });
            }
```

Continue

```
161
162             // Get student, decrypt, and return it
163             const student = await getStudentByStdId(studentId);
164             const decryptedStudent = await getDecryptedStudent(student);
165             return res.json(decryptedStudent);
166
167         } else if (req.role == "S") {
168
169             // Get student by requested user's id, and decrypt student id provided in URL
170             const student = await getStudentByUserId(req.userID);
171             const studentId = await Des.encrypt(queryParams.studentID);
172
173             // Check if user is NOT the student
174             if (studentId != student.studentId) {
175
176                 return res.status(403).send({
177                     Message: "You can only check info of your student",
178                 });
179             }
180
181             // Decrypt student, and return it
182             const decryptedStudent = await getDecryptedStudent(student);
183             return res.json(decryptedStudent);
184
185         } else if (req.role == "A") {
186
187             // Encrypt student id, get student, decrypt it, and return
188             const studentId = await Des.encrypt(req.params.studentID);
189             const student = await getStudentByStdId(studentId);
190             const decryptedStudent = await getDecryptedStudent(student);
191             return res.json(decryptedStudent);
192
193         } else {
194             return res.status(403).send({
195                 Message: "Only chair, parents, and admin can view student information",
196             });
197         }
198
199     } catch (error) {
200         console.log(error);
```

Add new attendance function controller

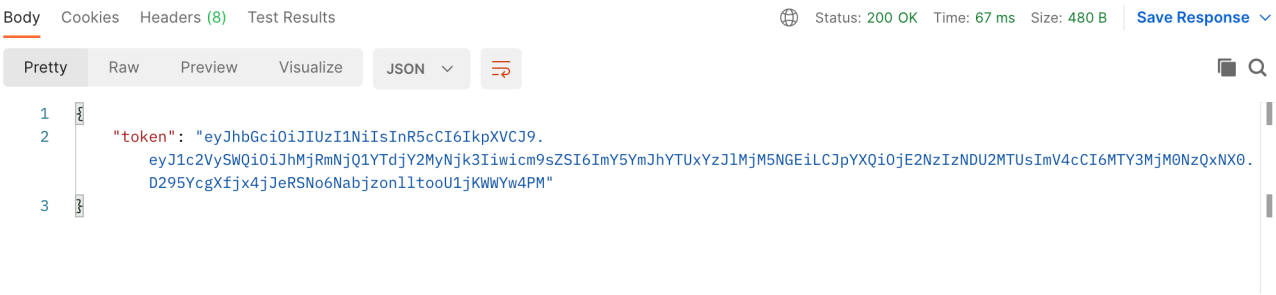
This function will be responsible for adding the attendance of the student. Only teacher and admin are allowed to add new attendance. If **teacher**, he/she should be teaching this course. Otherwise if **admin**, he will be authorized anyways.

```
const createAttendanceController = async(req, res) => {

    try {

        const body = req.body;
        const offeredCourseID = await Des.encrypt(req.params.offeredCourseID);

        // Check if the role is neither Admin nor Teacher
        if (req.role != 'A' && req.role != 'T') {
            return res.status(403).send({ 'Message': 'Only a teacher and admin can add new attendance' })
        }

        // Check if role is Teacher
        if (req.role == 'T') {

            const offeredCourse = await getOfferedCourseById(offeredCourseID);

            // Check if teacher does NOT teach this course
            if (offeredCourse.userId != req.userID) {
                return res.status(403).send({ 'Message': 'Only teacher teaches this course can add new attendance' })
            }

        }

        const student = await getStudentByStdId(await Des.encrypt(body.studentId));

        // Create new attendance for a student and encrypt it's information
        const attendance = await Attendance.create({
            offeredCourseCode: offeredCourseID,
            date: await Des.encrypt(req.body.date.toString()),
            isPresent: await Des.encrypt(body.isPresent),
            userId: student.userId,
        })
        await attendance.save()
        return res.status(200).send({ 'Message': 'Attendance added' })


    } catch (error) {
        console.log(error)
        return res.status(503).send({ 'Message': 'Something went wrong' })
    }
```

7. Conducted tests

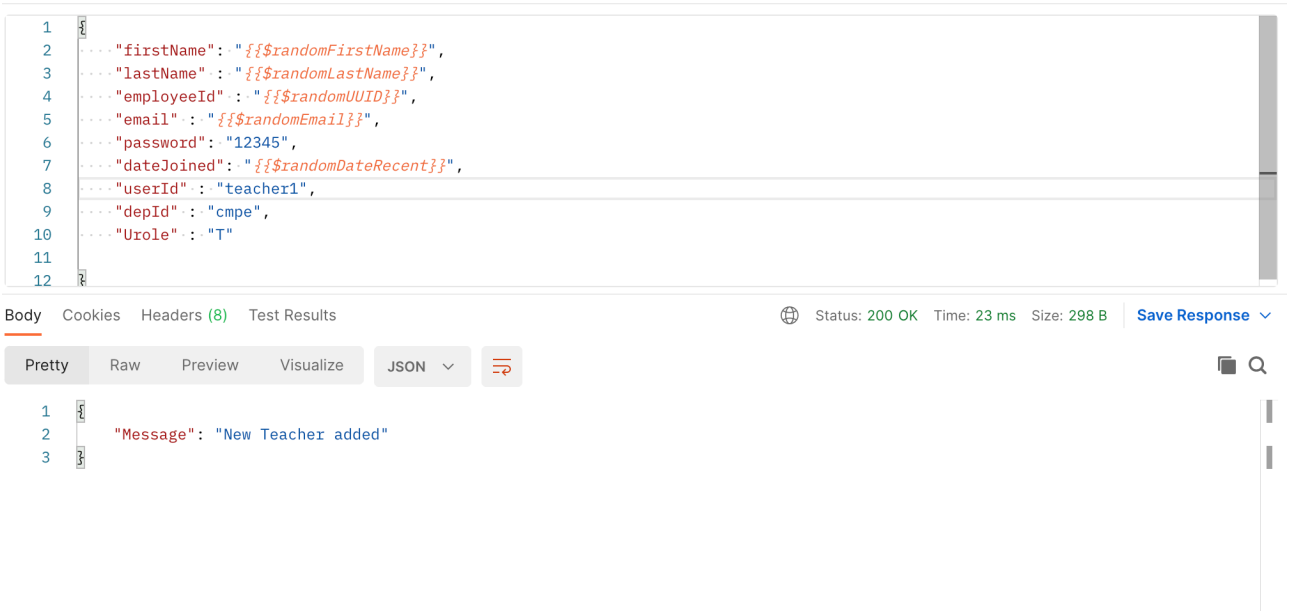
Test login as admin

As we can see in respond, new token generated



Test adding new student as admin

Test adding new teacher as admin



```
{
    "firstName": "{{$randomFirstName}}",
    "lastName": "{{$randomLastName}}",
    "employeeId": "{{$randomUUID}}",
    "email": "{{$randomEmail}}",
    "password": "12345",
    "dateJoined": "{{$randomDateRecent}}",
    "userId": "teacher1",
    "depId": "cmpe",
    "Urole": "T"

}
```

Body Cookies Headers (8) Test Results — Status: 200 OK Time: 23 ms Size: 298 B Save Response

Pretty Raw Preview Visualize JSON

```
{
    "Message": "New Teacher added"
}
```

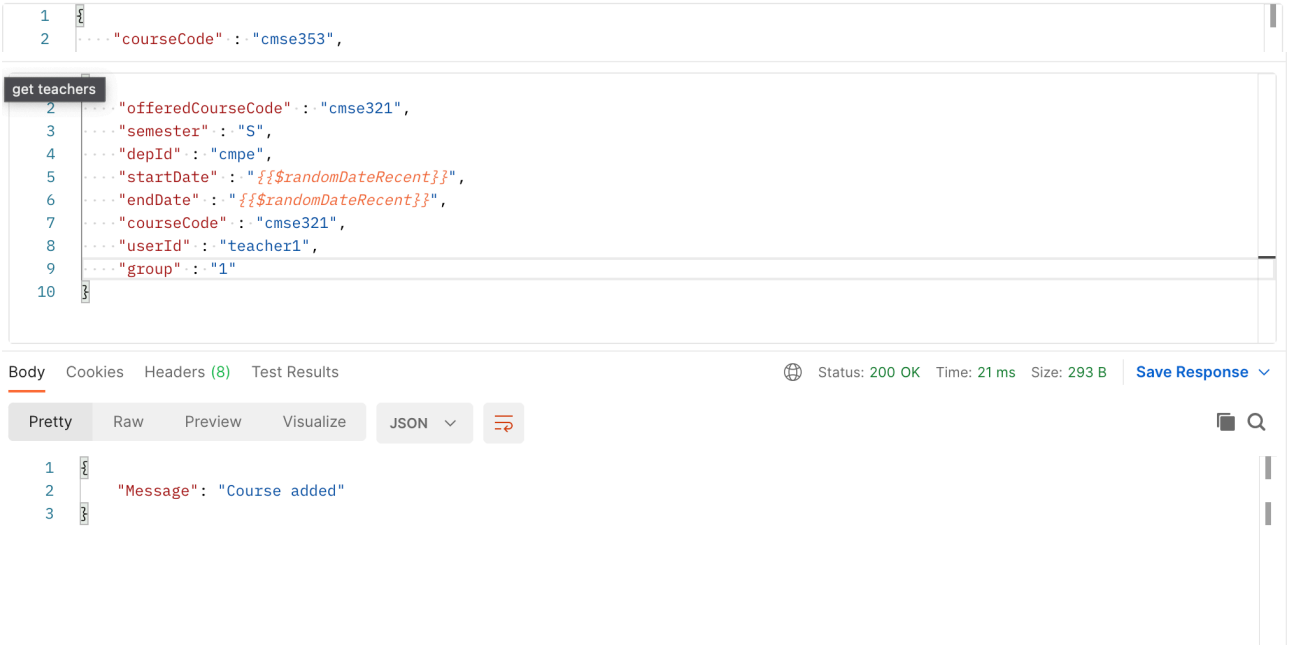
Test getting list of students

Body Cookies Headers (8) Test Results — Status: 200 OK Time: 12 ms Size: 1.51 KB Save Response

Pretty Raw Preview Visualize JSON


```
[
    {
        "userId": "student3",
        "email": "Rafaela7@yahoo.com",
        "firstName": "Shannon",
        "lastName": "Thiel",
        "dateJoined": "Thu Dec 29 2022 16:18:29 GMT+0200 (GMT+02:00)",
        "lastLogin": null,
        "dateOfBirth": null,
        "studentId": "2a513728-2fef-4943-b9c9-7cb72e26f26a",
        "currentCredits": "0",
        "pastCredits": "0",
        "CGPA": "0",
        "GPA": "0",
        "depId": "eeee"
    },
    {
        "userId": "student2",
        "email": "Andreane.Nikolaus@gmail.com",
        "firstName": "Anahi",
        "lastName": "Olson",
```

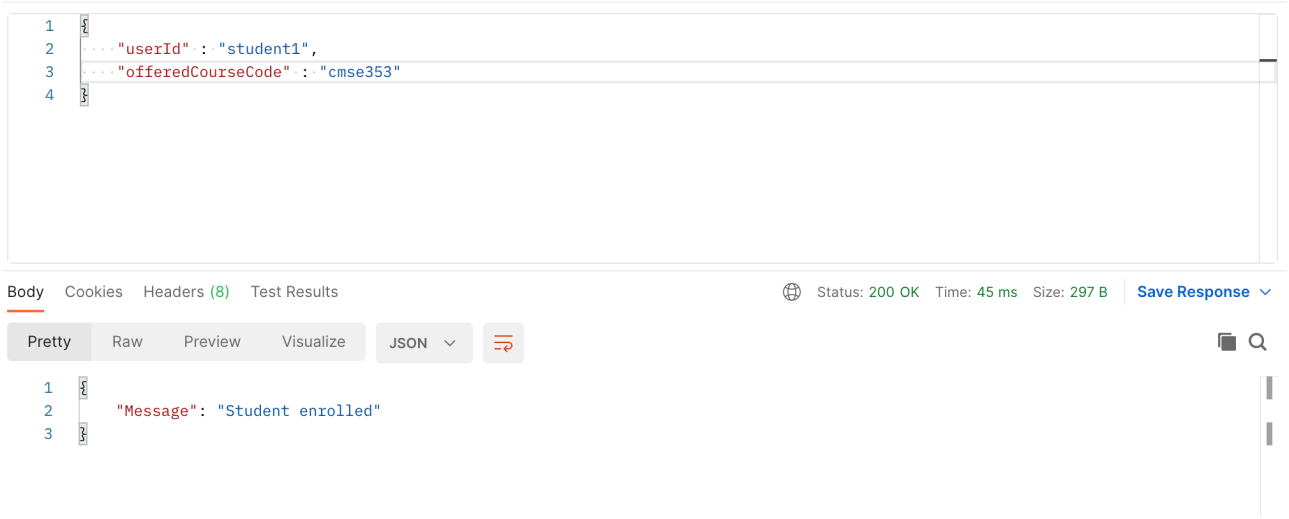

Test adding new course



Test offer new course as a teacher

Test getting list of offered courses as teacher

Test enrolling to an offered course as a student



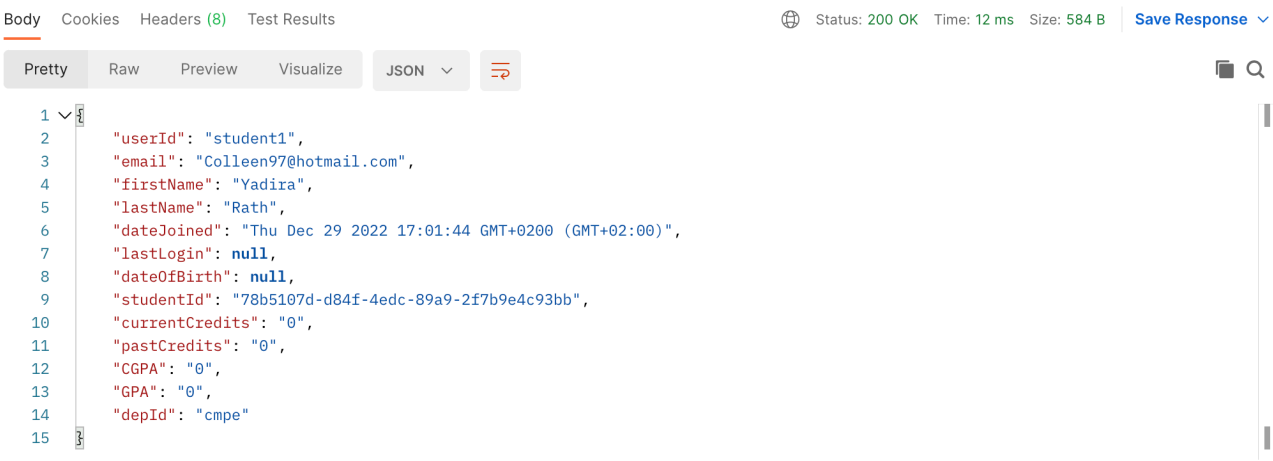
```
{
    "userId" : "student1",
    "offeredCourseCode" : "cmse353"
}
```

Body Cookies Headers (8) Test Results — Status: 200 OK Time: 45 ms Size: 297 B Save Response

Pretty Raw Preview Visualize JSON

```
{
    "Message": "Student enrolled"
}
```

Test getting student info as chair and parent



Body Cookies Headers (8) Test Results — Status: 200 OK Time: 12 ms Size: 584 B Save Response

Pretty Raw Preview Visualize JSON


```
{
    "userId": "student1",
    "email": "Colleen97@hotmail.com",
    "firstName": "Yadira",
    "lastName": "Rath",
    "dateJoined": "Thu Dec 29 2022 17:01:44 GMT+0200 (GMT+02:00)",
    "lastLogin": null,
    "dateOfBirth": null,
    "studentId": "78b5107d-d84f-4edc-89a9-2f7b9e4c93bb",
    "currentCredits": "0",
    "pastCredits": "0",
    "CGPA": "0",
    "GPA": "0",
    "depId": "cmpe"
}
```

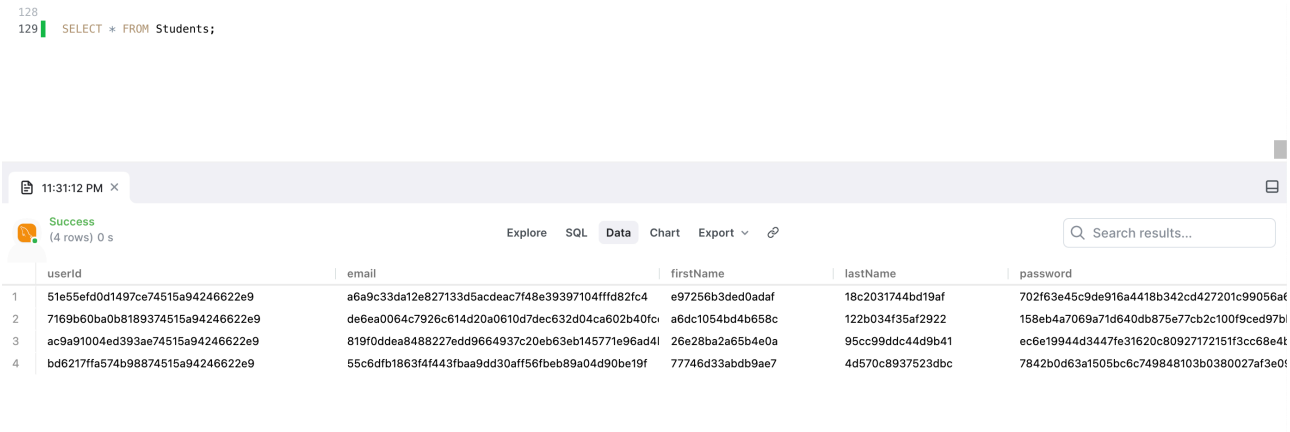

Test adding new attendance as a teacher

```
{
    "date" : "{{$randomDateRecent}}"
}
```

```
[
    {
        "CourseCode": "cmse321",
        "date": "Thu Dec 29 2022 04:44:08 GMT+0200 (GMT+02:00)",
        "isPresent": "true",
        "studentId": "78b5107d-d84f-4edc-89a9-2f7b9e4c93bb"
    },
    {
        "CourseCode": "cmse321",
        "date": "Thu Dec 29 2022 06:06:46 GMT+0200 (GMT+02:00)",
        "isPresent": "false",
        "studentId": "78b5107d-d84f-4edc-89a9-2f7b9e4c93bb"
    }
]
```

Test getting list of student attended to a course as a teacher

Test student data encrypted in database



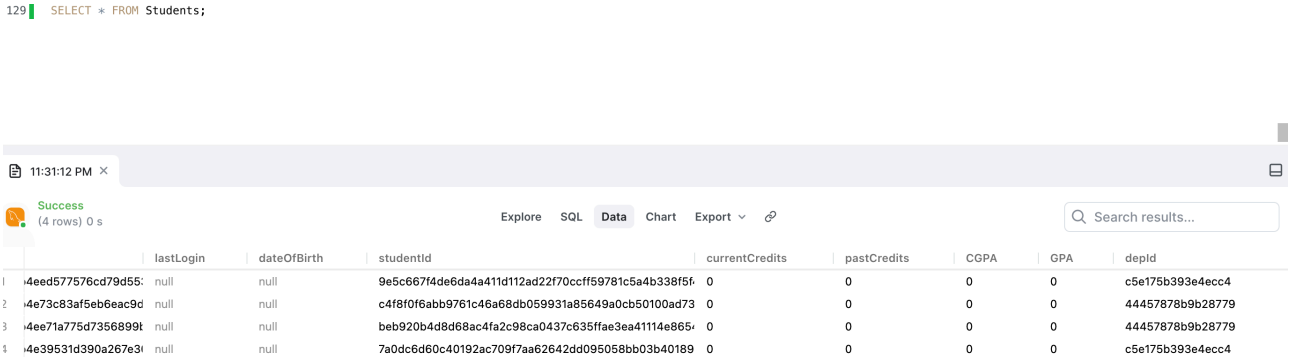
```
SELECT * FROM Students;
```

11:31:12 PM

Success
(4 rows) 0 s

Explore | SQL | Data | Chart | Export | Search results...

	userId	email	firstName	lastName	password
1	51e55efd0d1497ce74515a94246622e9	a6a9c33da12e827133d5acdeac7f48e39397104fffd82fc4	e97256b3ded0adaf	18c2031744bd19af	702f63e45c9de916a4418b342cd427201c99056a6
2	7169b60ba0b8189374515a94246622e9	de6ea0064c7926c614d20a0610d7dec632d04ca602b40fc	a6dc1054bd4b658c	122b034f35af2922	158eb4a7069a71d640db875e77cb2c100f9ced97b
3	ac9a91004ed393ae74515a94246622e9	819f0ddea8488227edd9664937c20eb63eb145771e96ad4	26e28ba2a65b4e0a	95cc99ddc44d9b41	ec6e19944d3447fe31620c80927172151f3cc68e4
4	bd6217ffa574b98874515a94246622e9	55c6dfb1863f4f443fbaa9dd30aff56fbeb89a04d90be19f	77746d33abdb9ae7	4d570c8937523dbc	7842b0d63a1505bc6c749848103b0380027af3e0



```
SELECT * FROM Students;
```

11:31:12 PM

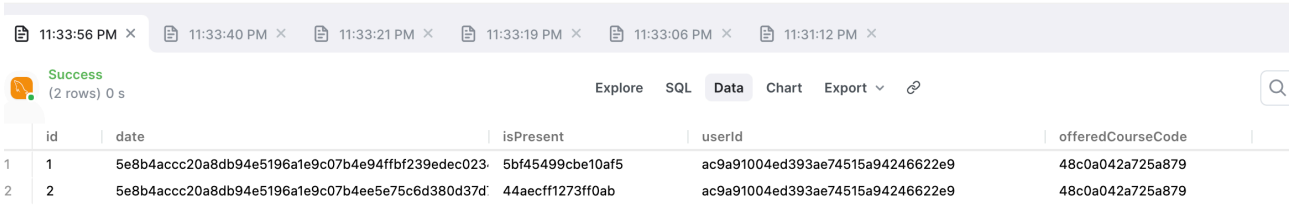
Success
(4 rows) 0 s

Explore | SQL | Data | Chart | Export | Search results...

		lastLogin	dateOfBirth	studentId	currentCredits	pastCredits	CGPA	GPA	depId
1	4eed577576cd79d55	null	null	9e5c667f4de6da4a411d112ad22f70ccff59781c5a4b338f5f	0	0	0	0	c5e175b393e4ecc4
2	4e73c83af5eb6eac9d	null	null	c4f8f0f6abb9761c46a68db059931a85649a0cb50100ad73	0	0	0	0	44457878b9b28779
3	4ee71a775d7356899t	null	null	beb920b4d8d68ac4fa2c98ca0437c635ffae3ea41114e865	0	0	0	0	44457878b9b28779
4	4e39531d390a267e3	null	null	7a0dc6d60c40192ac709f7aa62642dd095058bb03b40189	0	0	0	0	c5e175b393e4ecc4

Test attendance data is encrypted in database

```
129  SELECT * FROM Attendances;
```



11:33:56 PM | 11:33:40 PM | 11:33:21 PM | 11:33:19 PM | 11:33:06 PM | 11:31:12 PM

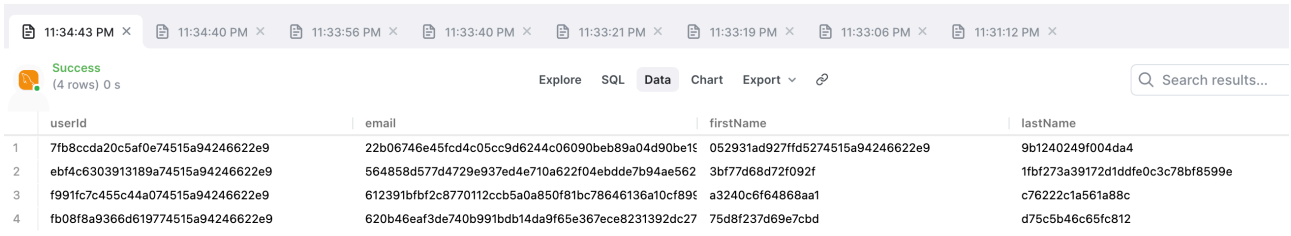
Success
(2 rows) 0 s

Explore | SQL | Data | Chart | Export

	id	date	isPresent	userId	offeredCourseCode
1	1	5e8b4accc20a8db94e5196a1e9c07b4e94ffbf239edec023	5bf45499cbe10af5	ac9a91004ed393ae74515a94246622e9	48c0a042a725a879
2	2	5e8b4accc20a8db94e5196a1e9c07b4ee5e75c6d380d37d	44aecff1273ff0ab	ac9a91004ed393ae74515a94246622e9	48c0a042a725a879

Test teachers data is encrypted in database

```
129  SELECT * FROM teachers;
```



11:34:43 PM | 11:34:40 PM | 11:33:56 PM | 11:33:40 PM | 11:33:21 PM | 11:33:19 PM | 11:33:06 PM | 11:31:12 PM

Success
(4 rows) 0 s

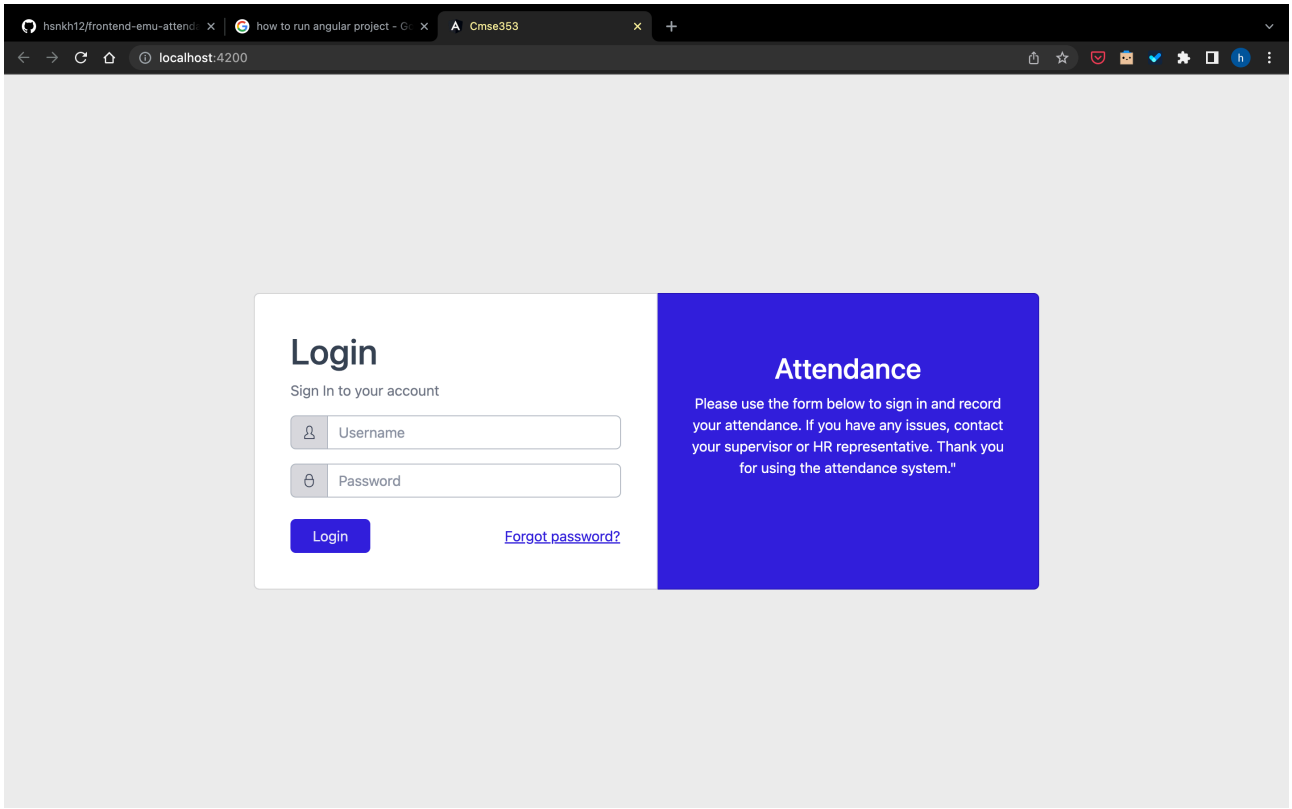
Explore | SQL | Data | Chart | Export

Search results...

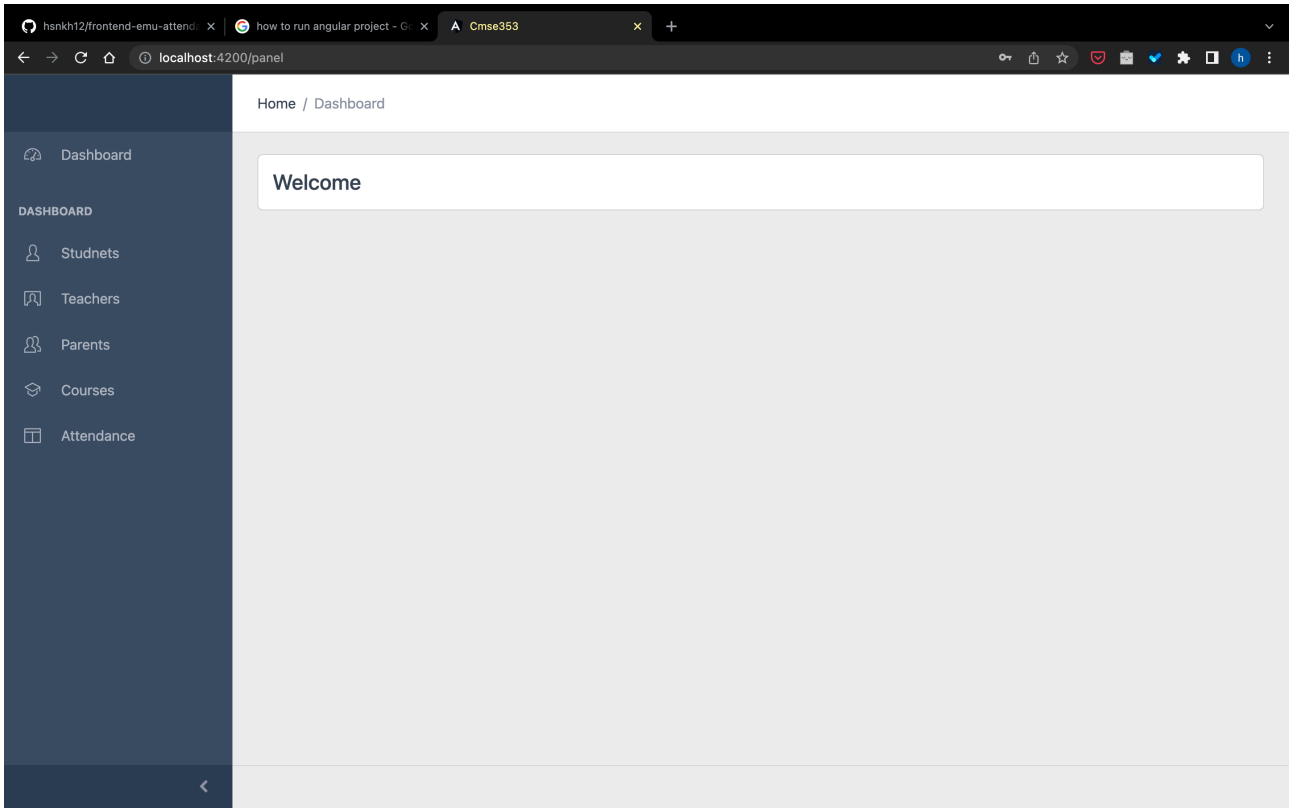
	userId	email	firstName	lastName
1	7fb8ccda20c5af0e74515a94246622e9	22b06746e45fcd4c05cc9d6244c06090beb89a04d90be19	052931ad927ffd5274515a94246622e9	9b1240249f004da4
2	ebf4c6303913189a74515a94246622e9	564858d577d4729e937ed4e710a622f04ebdde7b94ae562	3bf77d68d72f092f	1fbf273a39172d1ddfe0c3c78bf8599e
3	f991fc7c455c44a074515a94246622e9	612391bfbf2c8770112ccb5a0a850f81bc78646136a10cf89	a3240c6f64868aa1	c76222c1a561a88c
4	fb08f8a9366d619774515a94246622e9	620b46eaf3de740b991bdb14da9f65e367ece8231392dc27	75d8f237d69e7cbd	d75c5b46c65fc812

8. User guide

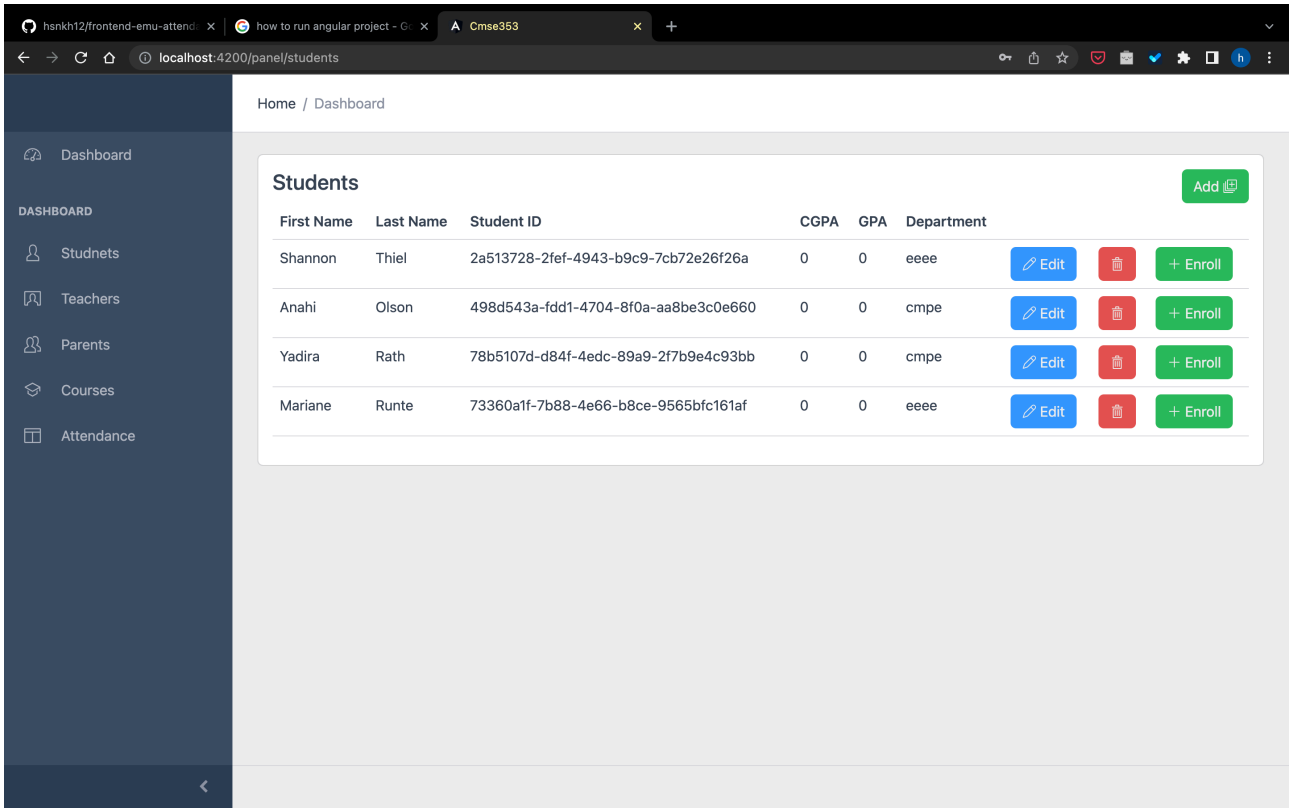
Login page



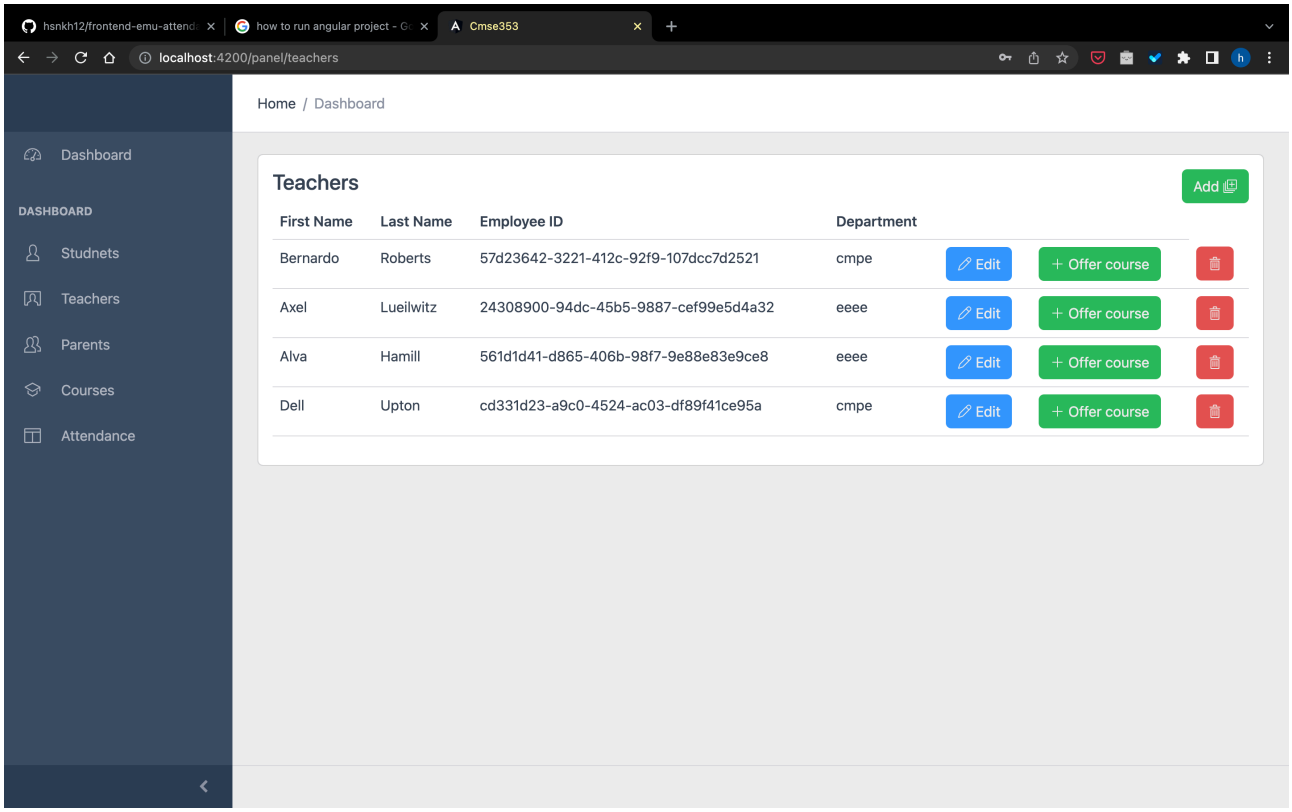
Admin point of view



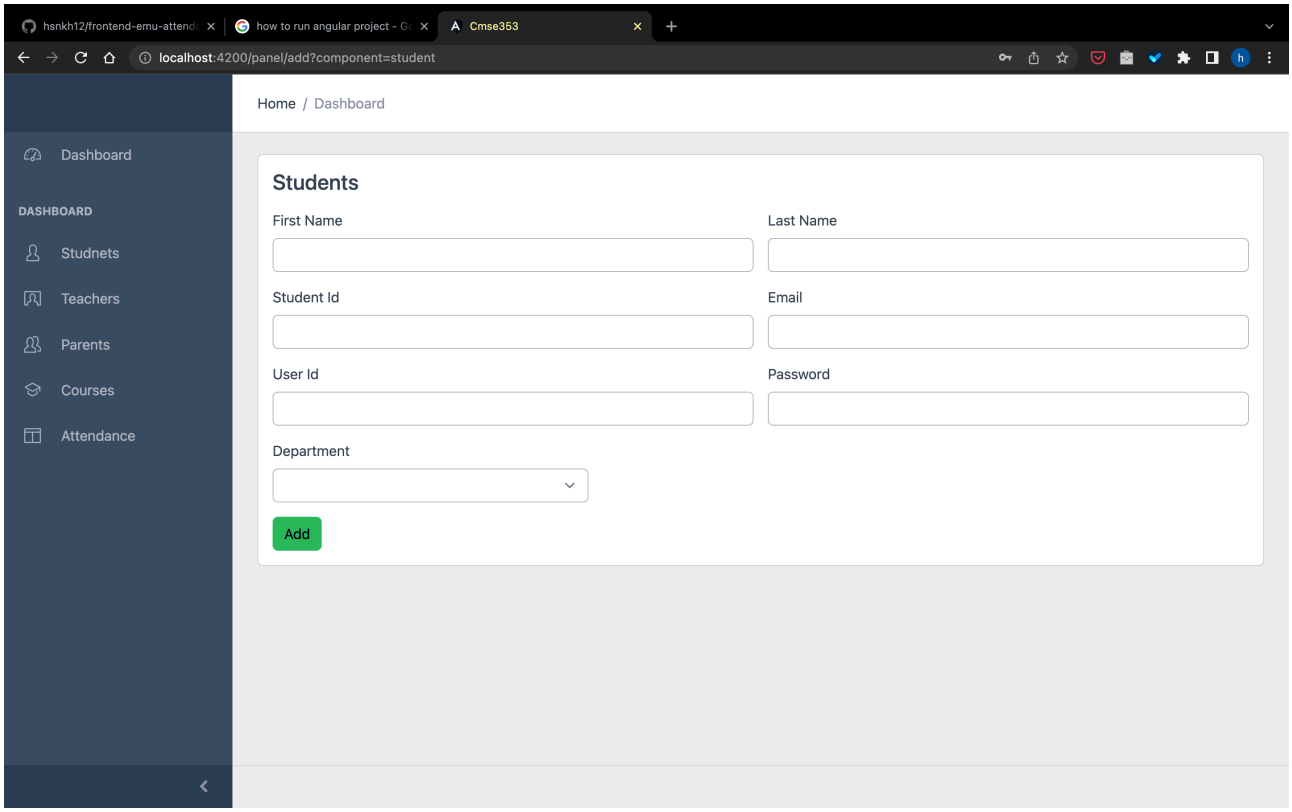
View students



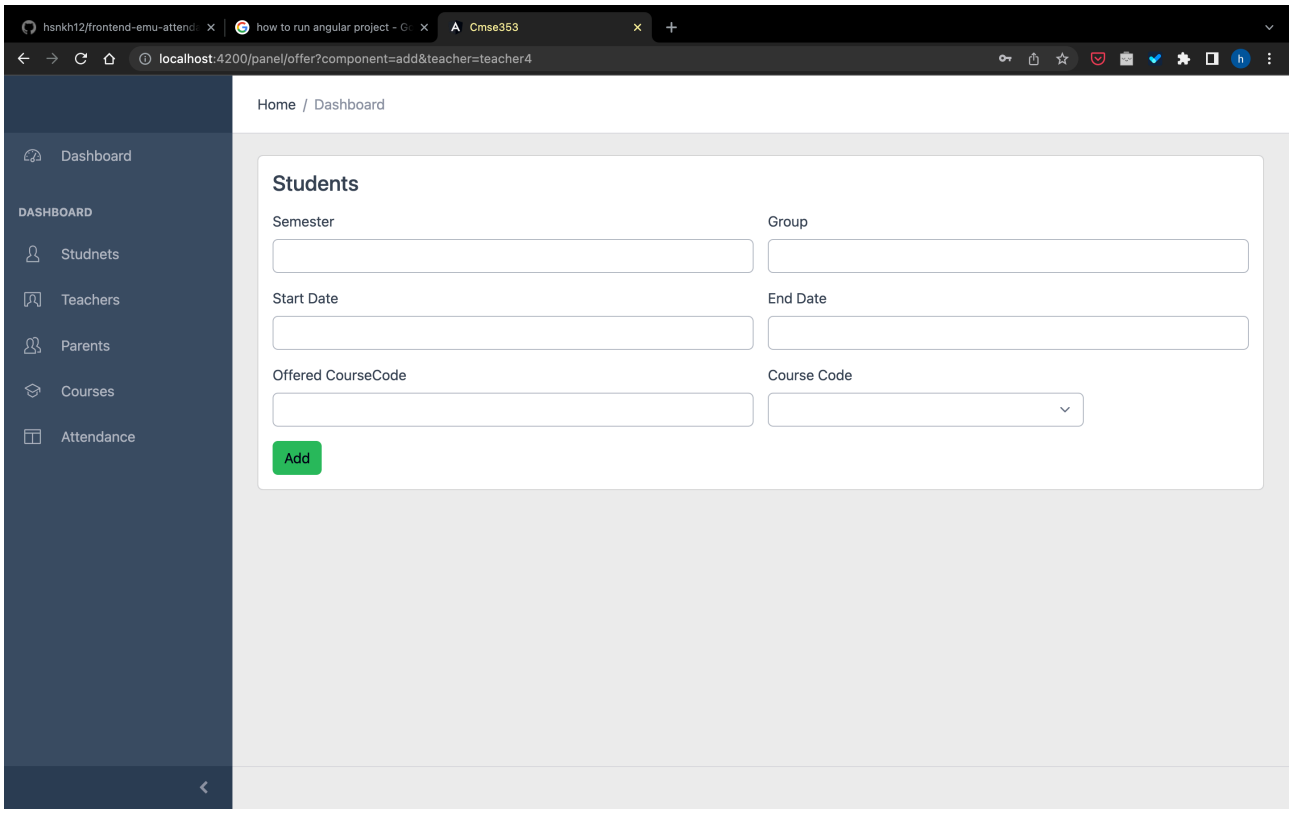
View teachers



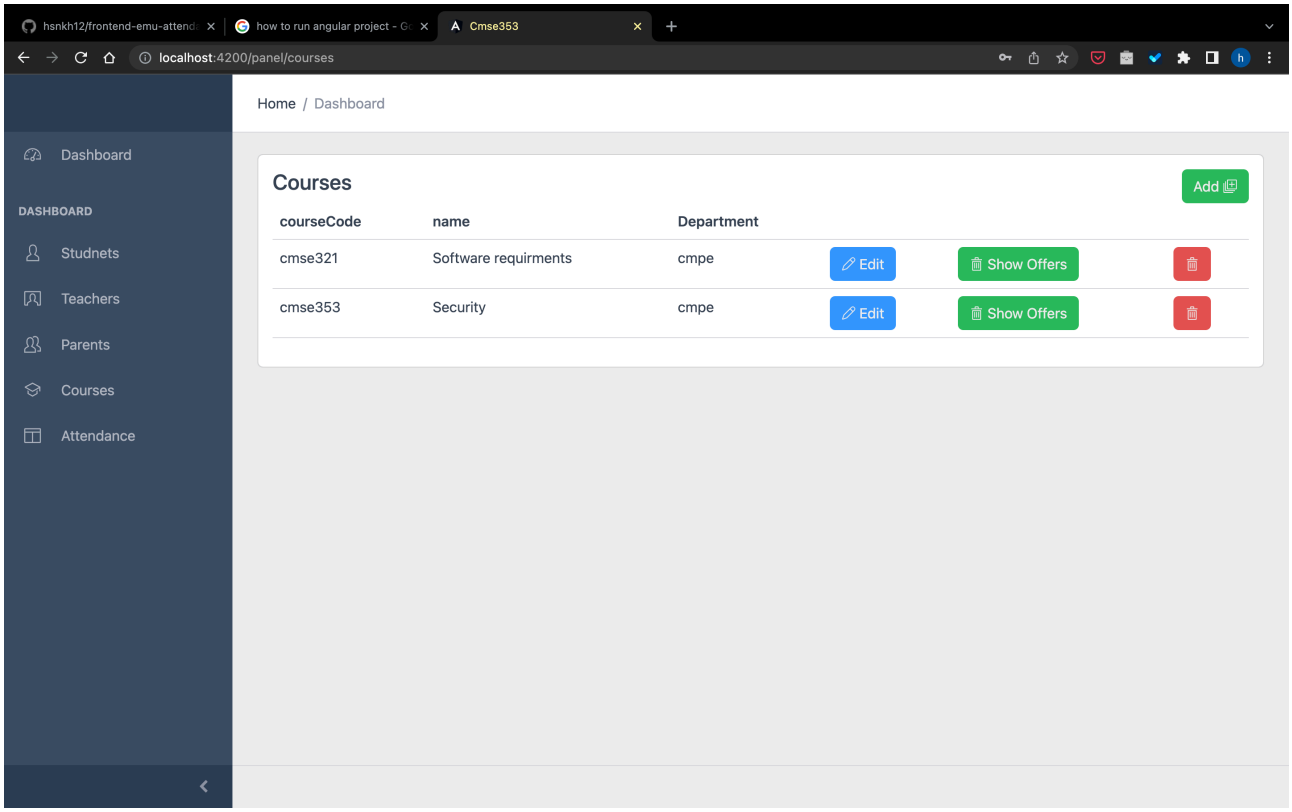
Add new student



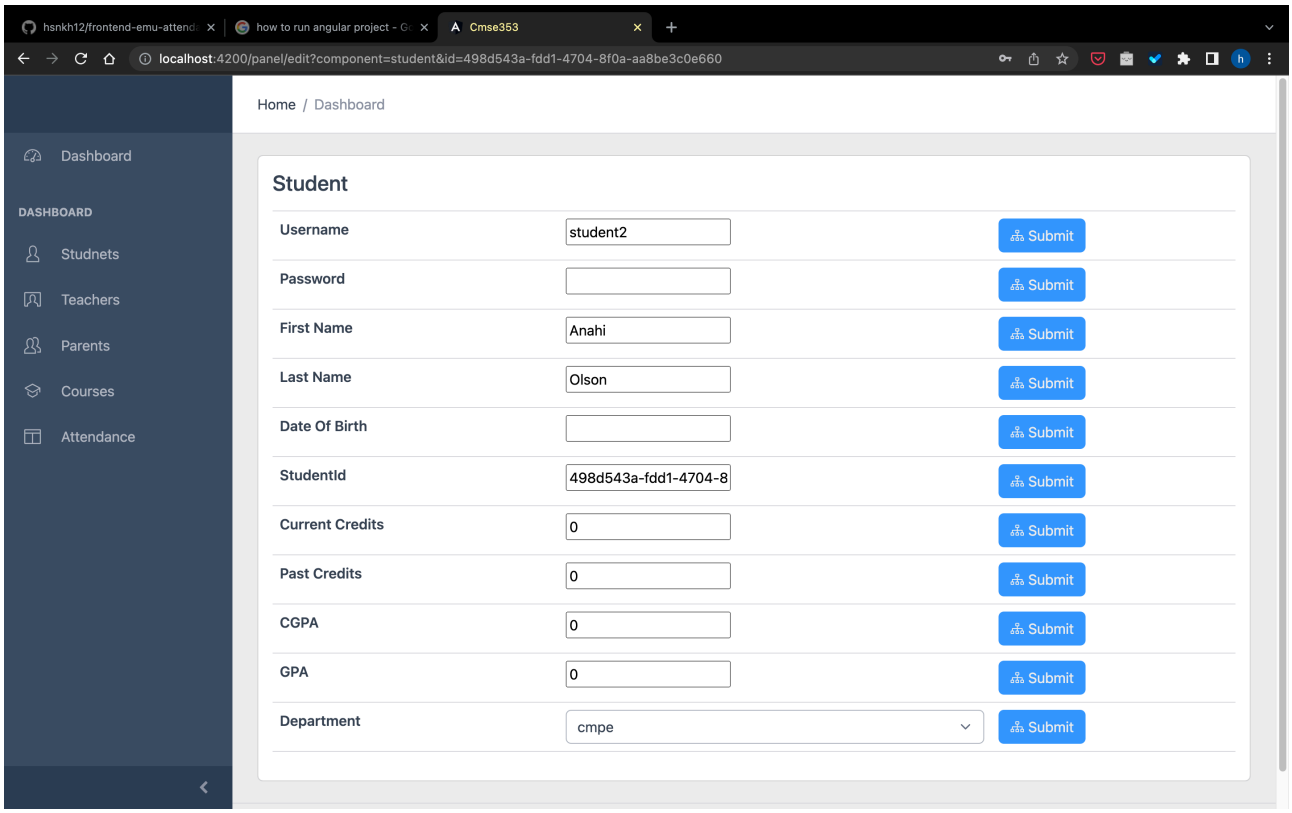
Add new teacher



View courses

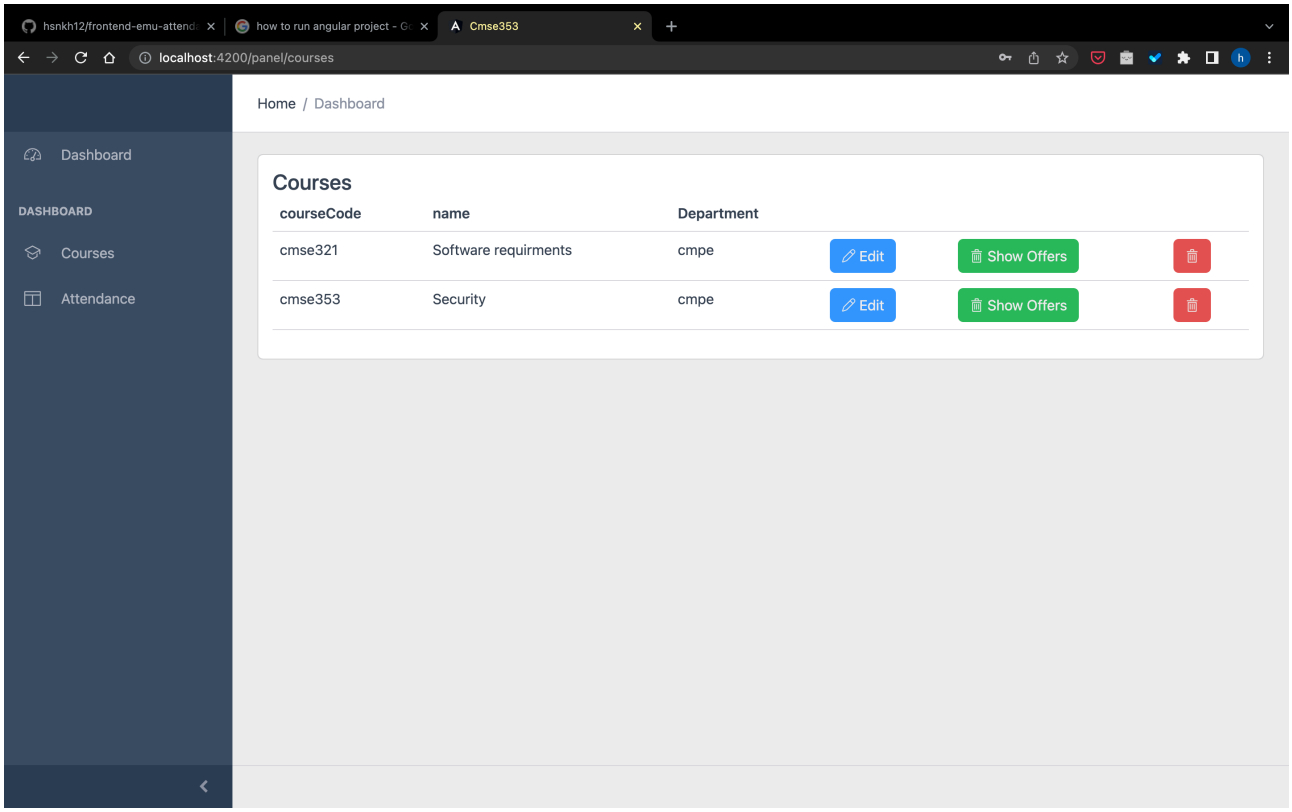


Update student

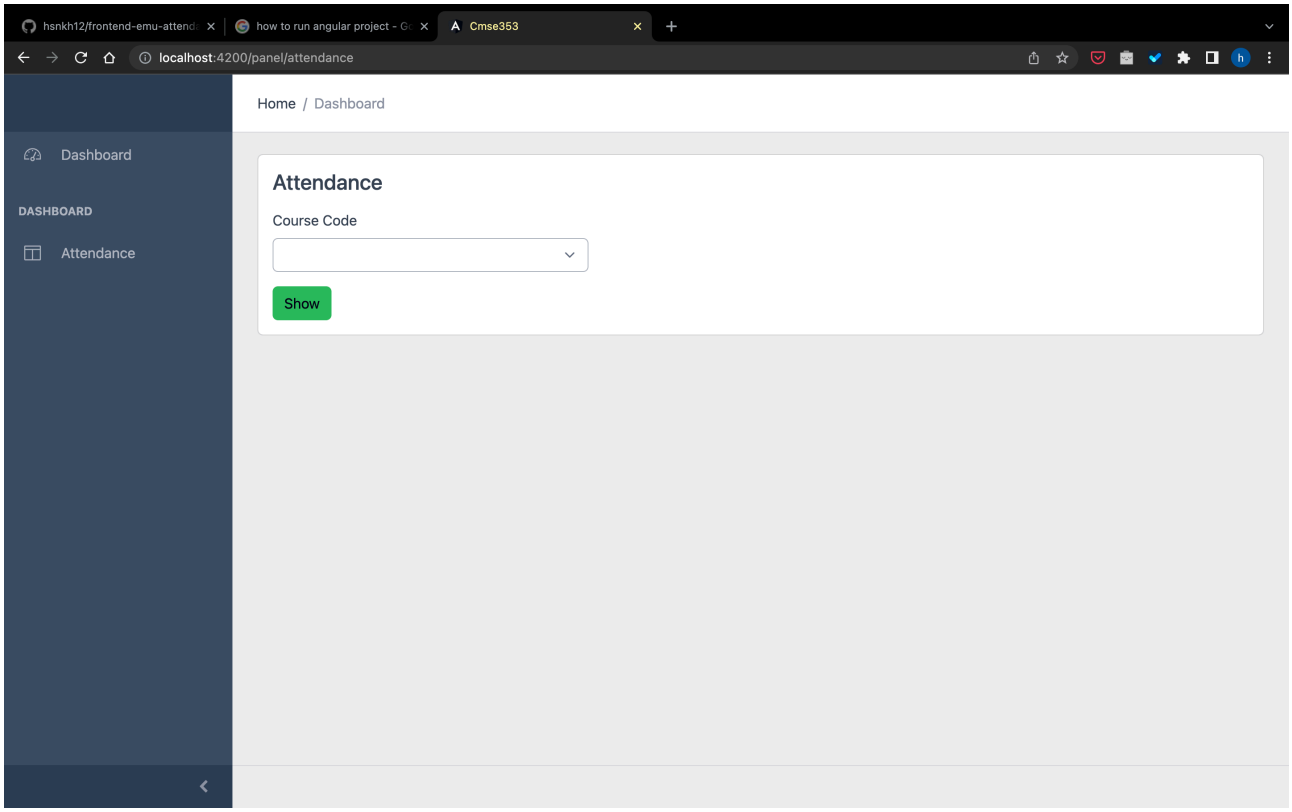


Teacher/chair point of view

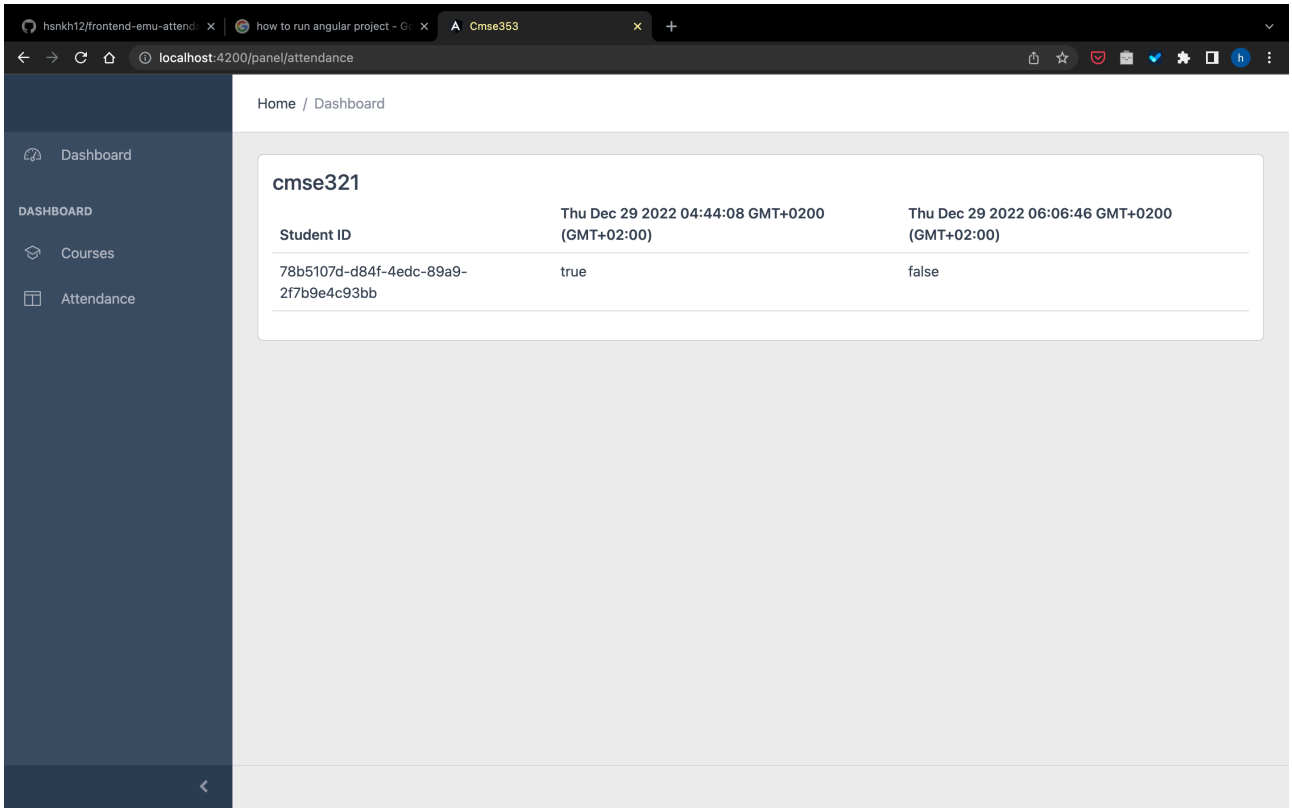
View courses



View Attendance (choose course code)

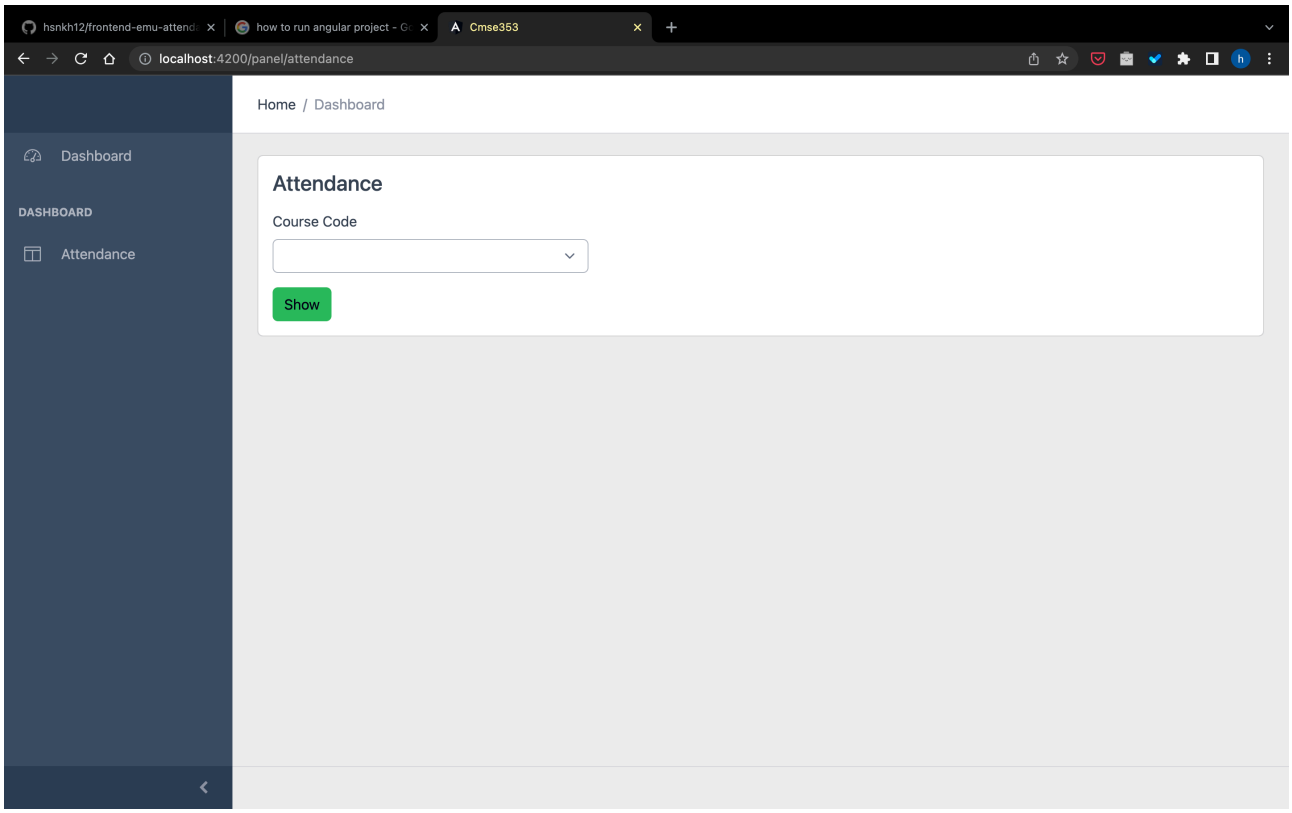


View Attendance

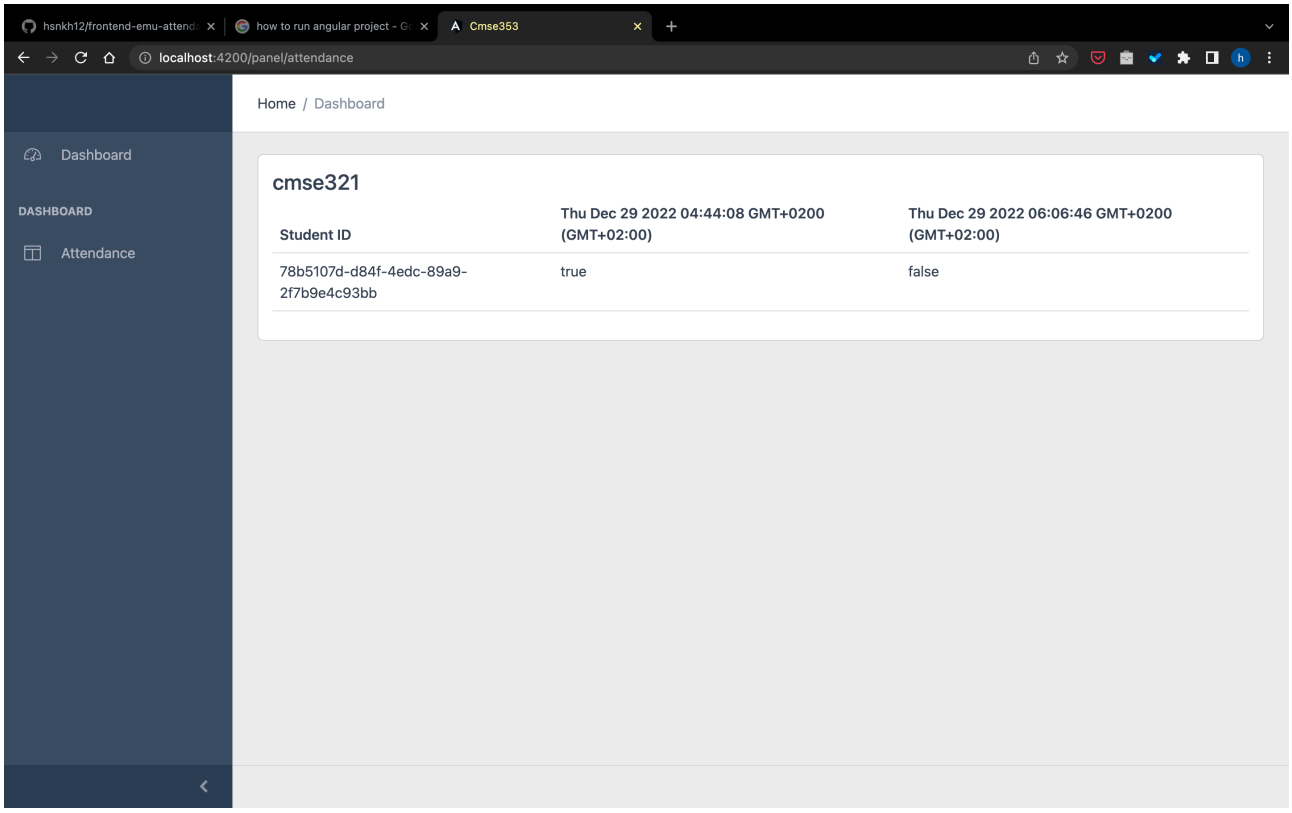


Student point of view

View Attendance (choose course code)



View Attendance of a course



9. Conclusion

The university attendance management system project has now been completed and has successfully been implemented at our institution. The system has greatly improved the efficiency and accuracy of attendance tracking, and has received positive feedback from both students and faculty.

One of the key features of the system is its ability to accurately track and record attendance for all classes, as well as generate reports for instructors and administrators. Students also have the ability to view their own attendance records, which has proven to be a valuable resource for staying on top of their academic progress.

Overall, the university attendance management system has been a resounding success and has helped to streamline many of the processes related to attendance tracking at our institution. We are confident that it will continue to be a valuable resource for students, faculty, and administrators alike.

10. How to use the system

RUN BACKEND

1- install nodeJs on your machine
2- run in command line 'cd backend'
3- run in command line 'npm i' to install requirements
4- run 'nodemon app.js' to run backend server on localhost
5- in you faced any issues run 'export NODE_OPTIONS=--openssl-legacy-provider' in the terminal

RUN FRONTEND

1- install nodeJs on your machine
2- un in command line 'cd frontend'
3- run in command line 'npm i' to install requirements
4- run 'npm start' to run the frontend