SchoolManager Documentation

Release alpha 1.0

ITUCSDB1932

December 27, 2019

CONTENTS

Team itucsdb1932

Members

- Muhammed Furkan Kamer
- Burak Bozdağ

SchoolManager is an application that enables schools to manage courses, exams, students, etc. more easily and systematically.

CHAPTER

ONE

USER GUIDE

1.1 Parts Implemented by Muhammed Furkan Kamer

1.1.1 New Users

To use this site, you must sign up for our application. If you are not signed in or signed up yet, you will probably see a page like this:

Fig. 1.1: The initial page for unsigned users. Sign up using the button and proceed to the next step.

1.1.2 Signing Up

When you click the *Sign Up* button, a form will appear in your browser. First, you must select a radio button for which account type do you want to sign up. Then a complete form will appear on screen according to your choice. For example, in below a *Manager* signs up and in addition to common boxes like *username*, *password*, etc there is an *experience year* box which must be

filled. You cannot enter anything other than a number, if you did it will warn you to enter a number. It can all be seen below:

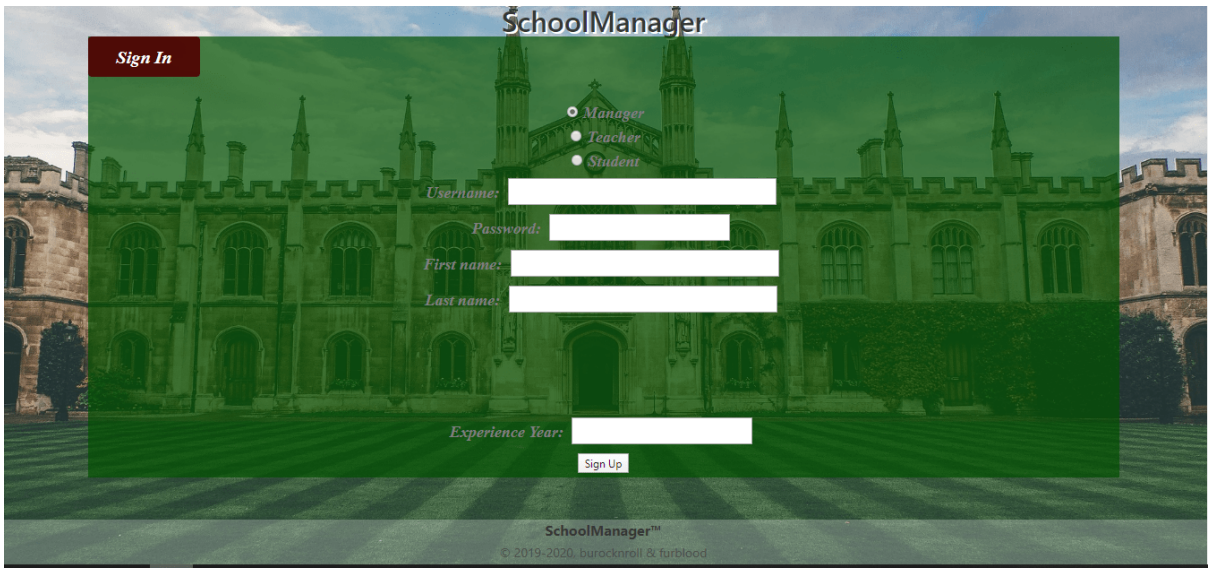

Fig. 1.2: Enter necessary information to boxes.

1.1.3 Signing In

When you click the *Sign In* button, a little form will appear in your browser. Type your username and password into these boxes and click *Sign In*.

Fig. 1.3: Enter your username and password to log in.

After logging in, you should see the home page for signed users.

Fig. 1.4: Home page for signed users.

1.1.4 Lectures

When you signed in successfully and see the homepage if you are a teacher or student you can route to Lectures page for lecture registry or creation.

Teachers' Lectures Page

When logged in as a teacher, you can create a lecture from the lectures page using select tags. You can select lecture name, time, day and building and quota for your new lecture. After selections, you should click submit to create a lecture. If any other lecture exists that overlaps, it redirects to the lectures page and warns you that there is another lecture that overlaps. Below we can see all these features.

Students' Lectures Page

When logged in as a student, a list of lectures appears on the screen. From that is every row has information about lectures and also a radio button which is used to select a lecture to register. Only one lecture can be selected at the same time. When you select a lecture, you should click the submit button. If no lecture of yours overlaps with these it registers successfully. Else it will show a warning about overlap.

1.1.5 Etudes

When you signed in successfully and see the homepage if you are a teacher or student you can also route to Etudes page for etudes registry or creation.

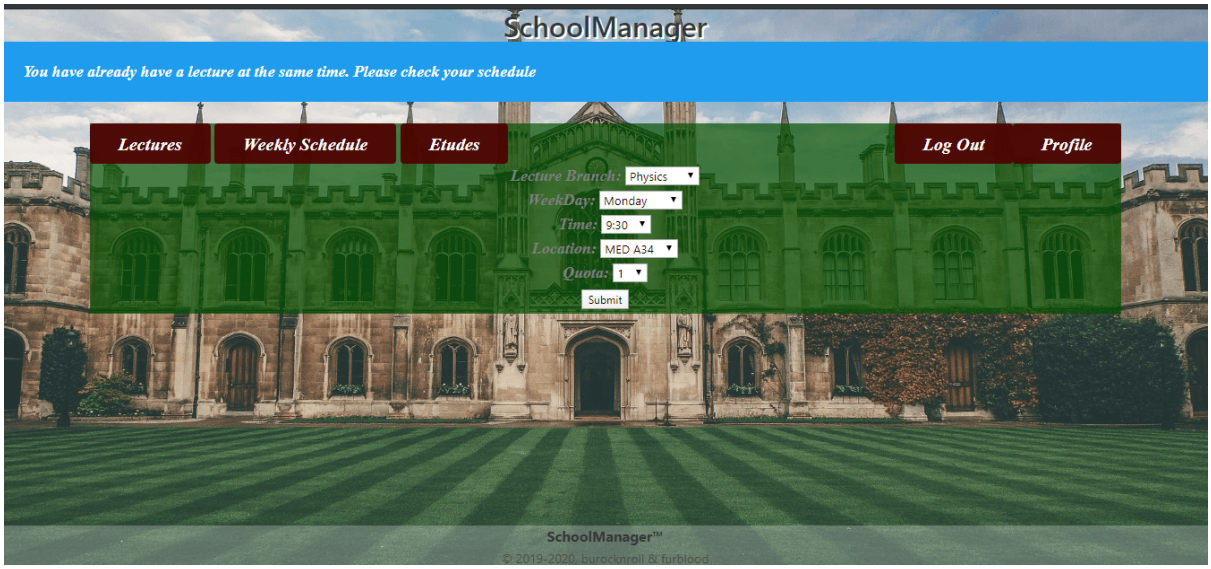

Fig. 1.5: Teachers' Lectures Page

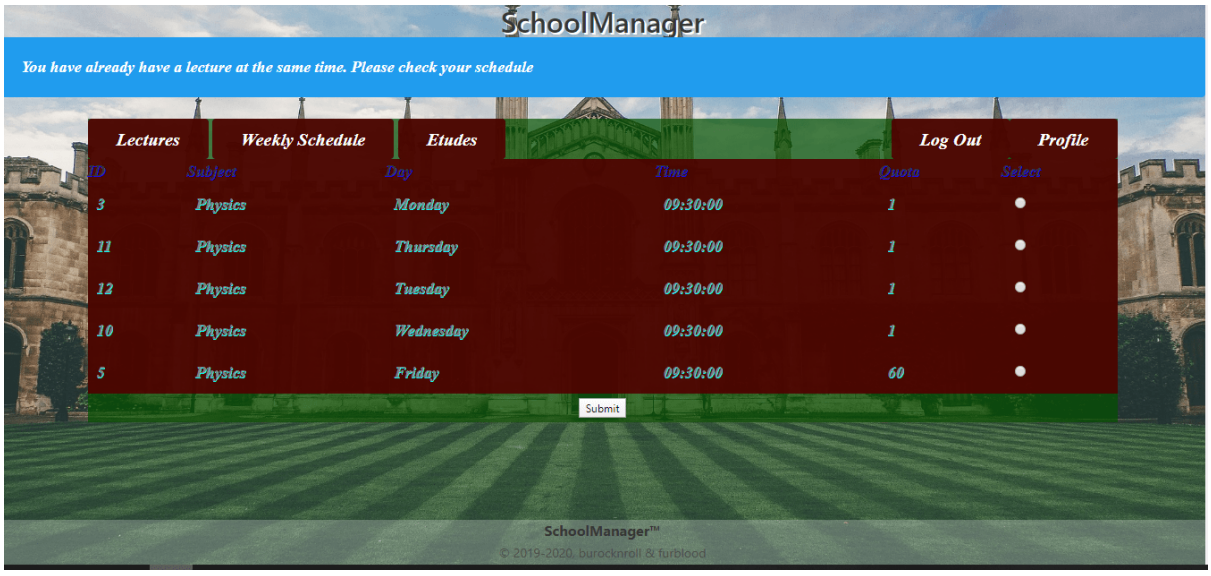

Fig. 1.6: Students' Lectures Page

Teachers' Etudes Page

When logged in as a teacher, you can create an etude from the etudes page using select tags. You can select etude subject, time, day and building and quota for your new etude. After selections, you should click submit to create an etude. If any other etude exists that overlaps, it redirects to the etudes page and warns you that there is another etude that overlaps. Below we can see all these features.

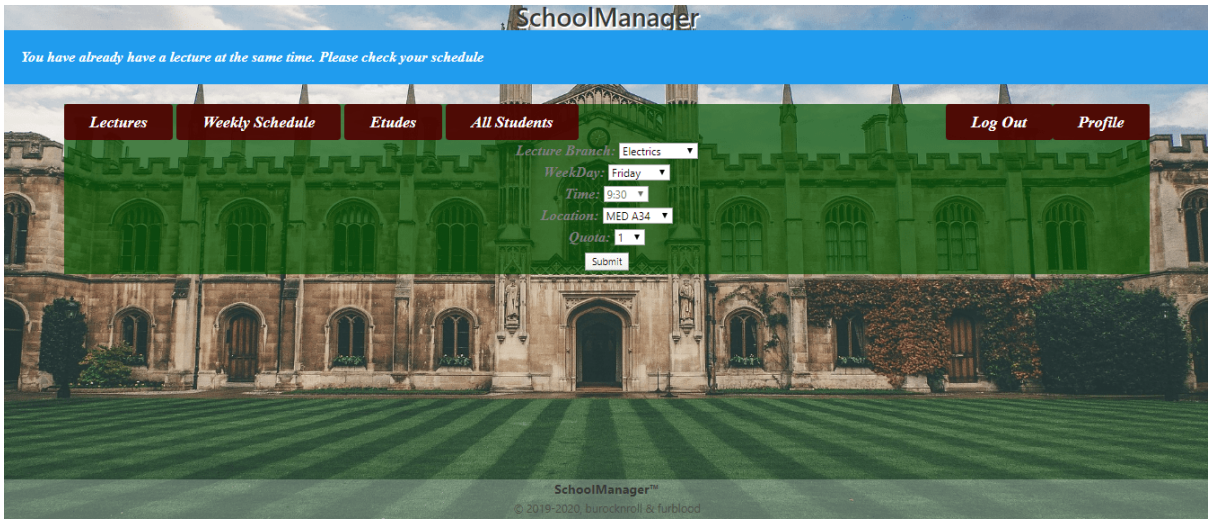

Fig. 1.7: Teachers' Etudes Page

Students' Etudes Page

When logged in as a student, a list of etudes appears on the screen. From that is every row has the information of etudes and also a radio button which is used to select an etude to register. Only one etude can be selected at the same time. When you select an etude, you should click the submit button. If no etude of yours overlaps with these it registers successfully. Else it will show a warning about overlap.

Fig. 1.8: Students' Etudes Page

1.1.6 Schedule

When you signed in successfully and see the homepage if you are a teacher or student you can also route to schedule a page and see your weekly schedule.

Teachers' Schedule Page

When logged in as a teacher, you can see your weekly schedule on the schedule page. Also, you can delete or update your lecture or etude in the schedule using an id that is given in the schedule table. You can update every information of the lecture other from id.

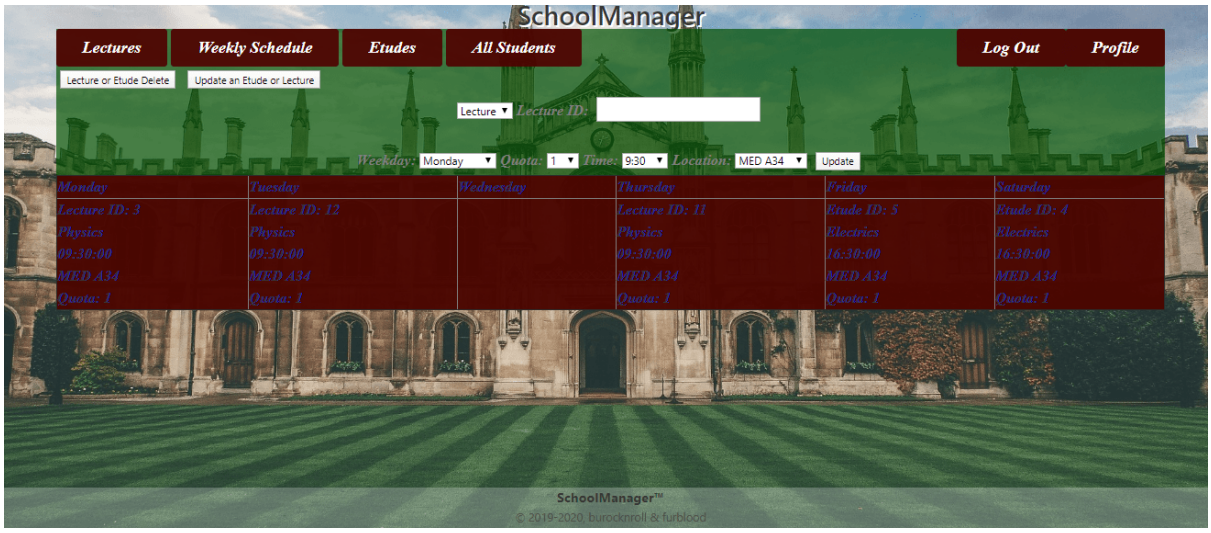

Fig. 1.9: Teachers' Schedule Page

Students' Schedule Page

When logged in as a student, you can see your weekly schedule on the schedule page. Also, you can delete your registry from the lecture or etude in the schedule using an id that is given in the schedule table. When you did it will be deleted from your schedule and number of enrolled students.

1.2 Parts Implemented by Burak Bozdağ

1.2.1 New Users

In order to use this site, you must sign up to our application. If you are not signed in or signed up yet, you will probably see a page like this:

1.2.2 Signing In

When you click the *Sign In* button, a little form will appear in your browser. Type your username and password into these boxes and click *Sign In*.

After logging in, you should see the home page for signed users.

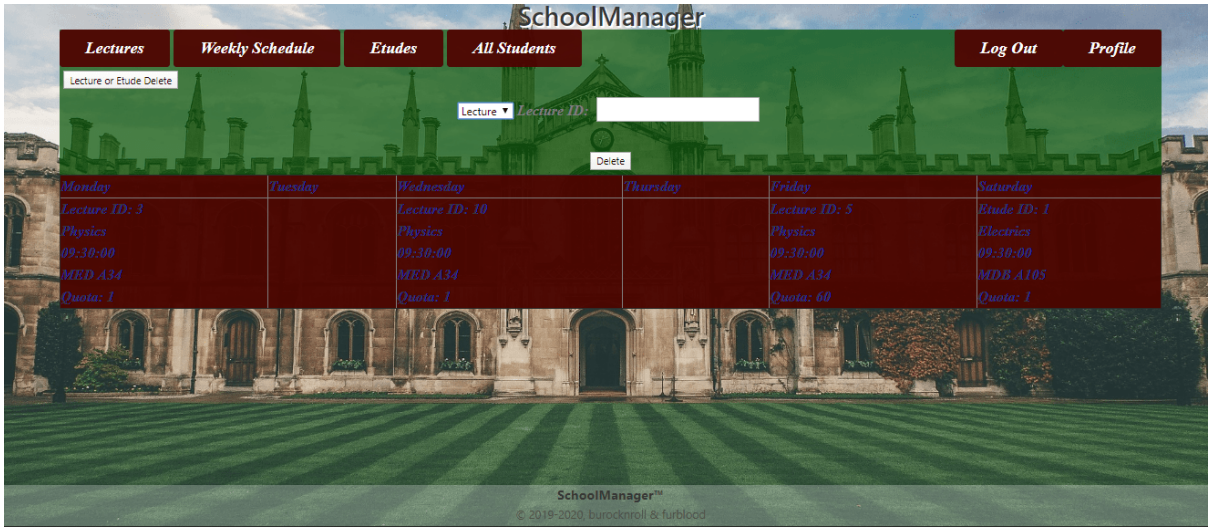

Fig. 1.10: Students' Schedule Page

Fig. 1.11: The initial page for unsigned users. Sign up using the button and proceed to the next step.

Fig. 1.12: Enter your username and password to log in.

Fig. 1.13: Home page for signed users.

1.2.3 Profile Actions

If you want to view your profile, you can click on the *Profile* button.

Fig. 1.14: The profile page.

Updating Profile

In order to update your profile, you should click on the *Update Your Profile* button.

Change attributes which you want and click on the *Update* button. You will be redirected to your profile page.

Updating Password

In order to update your password, you should click on the *Change Your Password* button.

Enter your old password once and your new password twice. You will be logged out and redirected to the log in page for security purposes.

Deleting Account

In order to delete your account, you should click on the *DELETE YOUR ACCOUNT!* button. You will see a pop-up message that confirms your action. After that, you will be logged out and redirected to the initial page.

1.2.4 Guides

Depending on your title, you can use various functions in the site. In next sections, the user guide will be divided based on user titles.

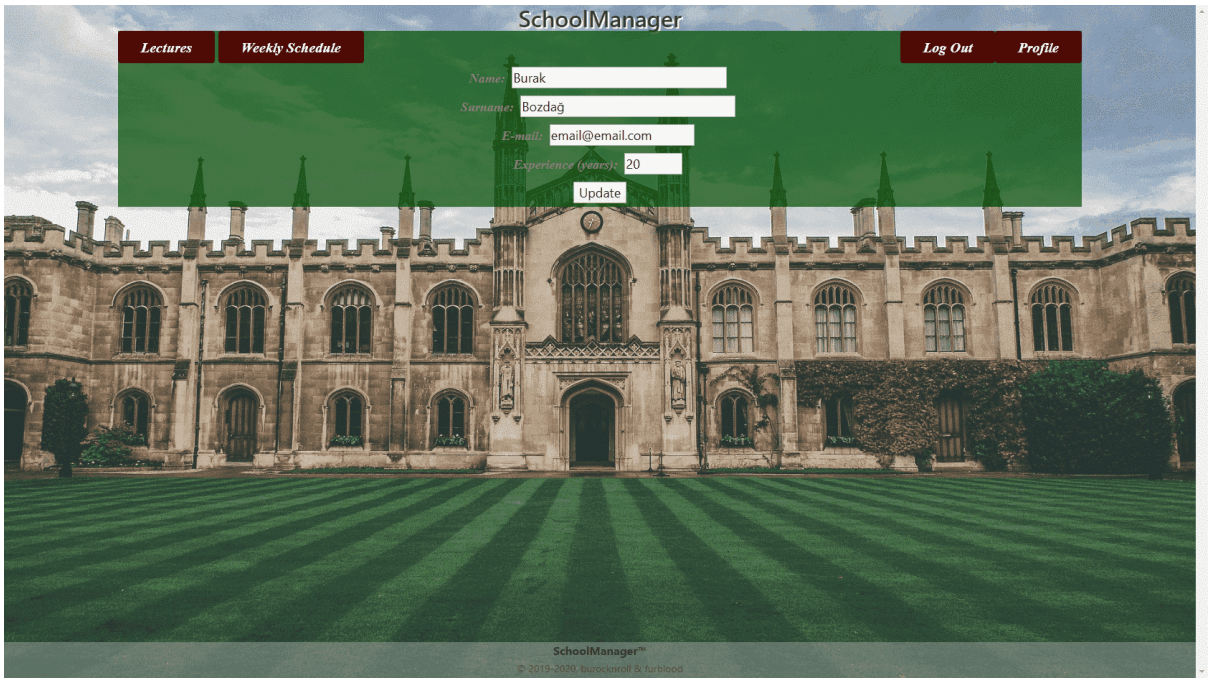

Fig. 1.15: The profile updating page.

Fig. 1.16: The page for password changes.

Managers' Guide

As a manager, you have the biggest authorization in this application. You can view all teachers, students. You can also grade students. These functions are explained in next sections.

Viewing All Teachers

If you want to view all teachers in the database, you should click on the *All Teachers* button.

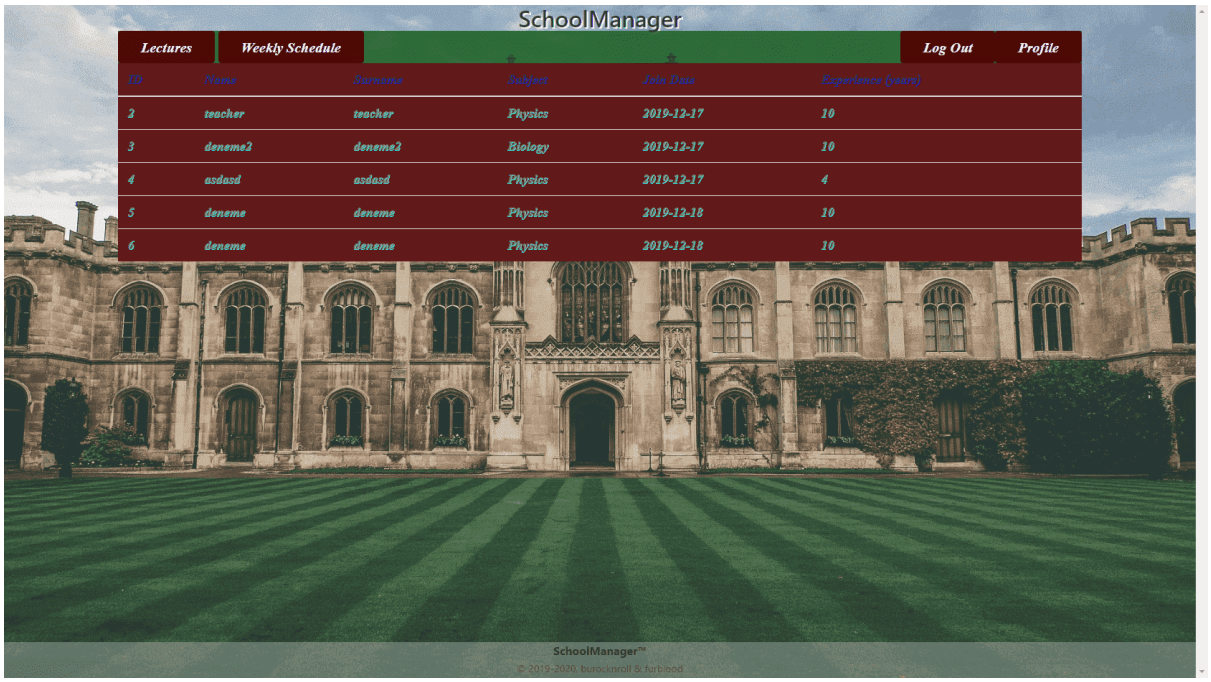

Fig. 1.17: All teachers in the database.

Viewing All Students

If you want to view all students in the database, you should click on the *All Students* button.

Grading Students

You can grade students using the previous page. You should write the student's ID into the box below, then click *Submit*. After that, you will see the profile of the student.

After entering a grade, click *Submit* and you will be redirected to the students page.

Teachers' Guide

As a teacher, you are responsible for giving lectures and etudes to your students. You can also grade students. I implemented the grading part, so in this page, you will see only the grading guide.

Viewing All Students

If you want to view all students in the database, you should click on the *All Students* button.

Fig. 1.18: All students in the database.

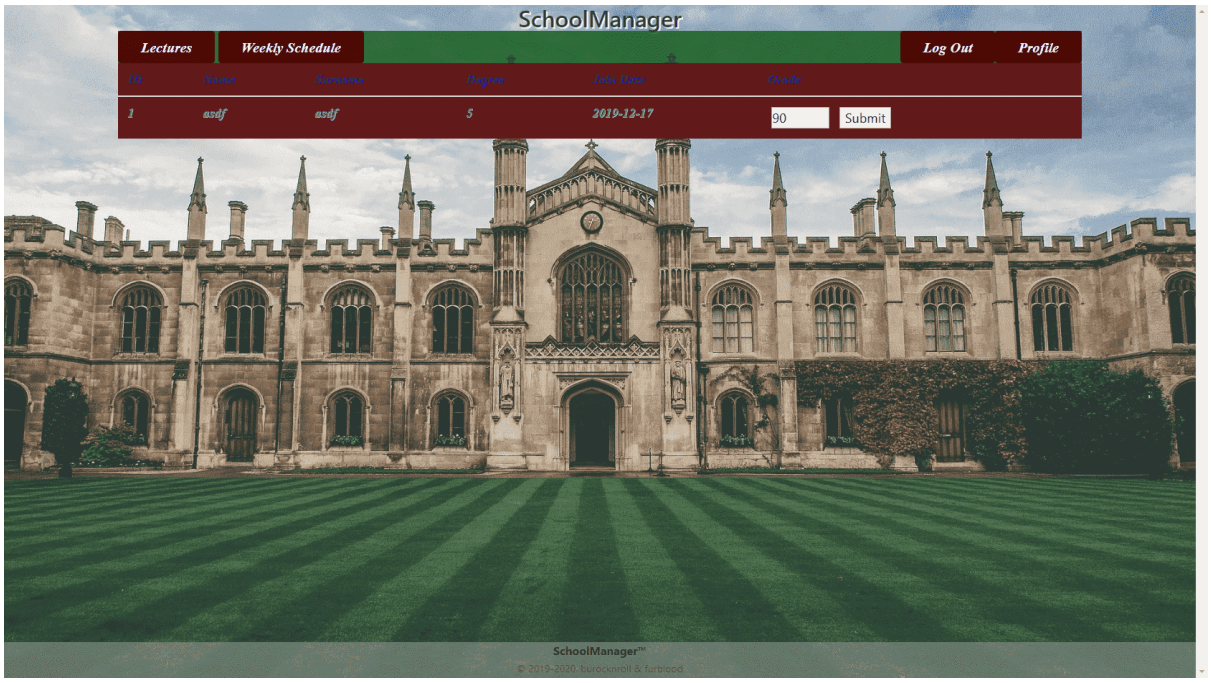

Fig. 1.19: The grading page.

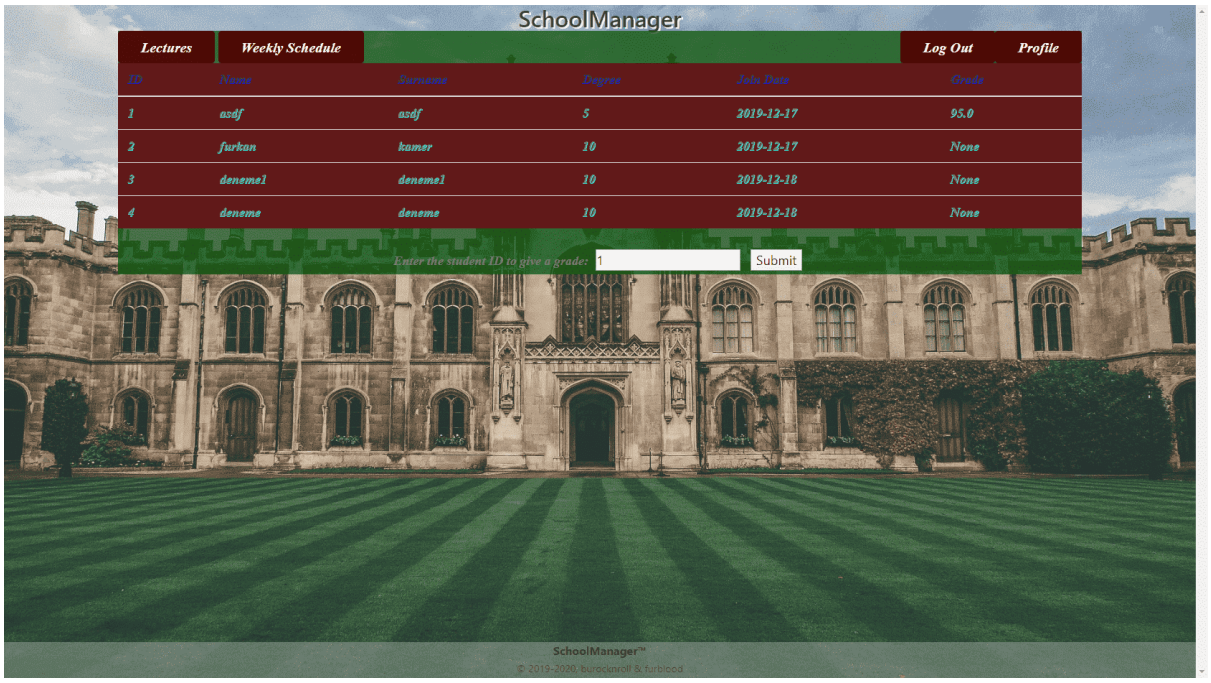

Fig. 1.20: All students in the database.

Grading Students

You can grade students using the previous page. You should write the student's ID into the box below, then click *Submit*. After that, you will see the profile of the student.

After entering a grade, click *Submit* and you will be redirected to the students page.

Students' Guide

As a student, you are responsible for taking lectures and etudes. You can also view your weekly schedule. I did not implemented lectures and etudes, so this guide is written in my partner's section.

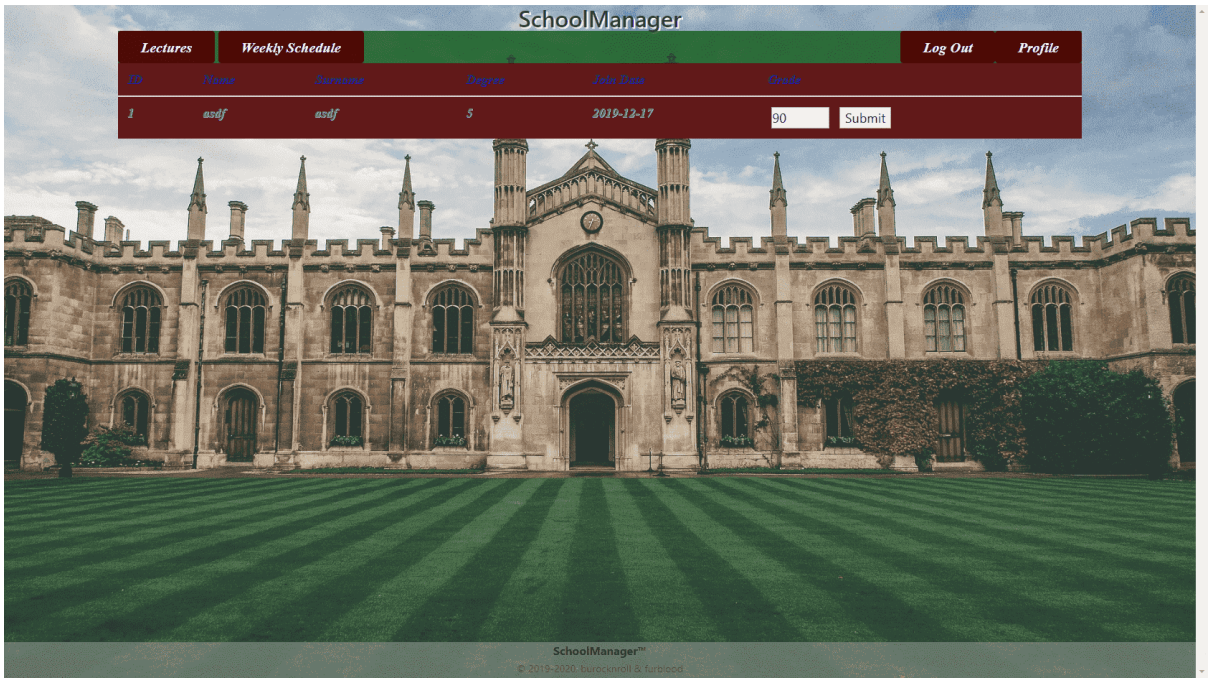

Fig. 1.21: The grading page.

CHAPTER

TWO

DEVELOPER GUIDE

2.1 Parts Implemented by Muhammed Furkan Kamer

2.1.1 Database Design

I implemented three tables and two extra tables in this project: *Users*, *Lectures* and *Etudes* and *registeredstudents*,*building*.

In my tables;

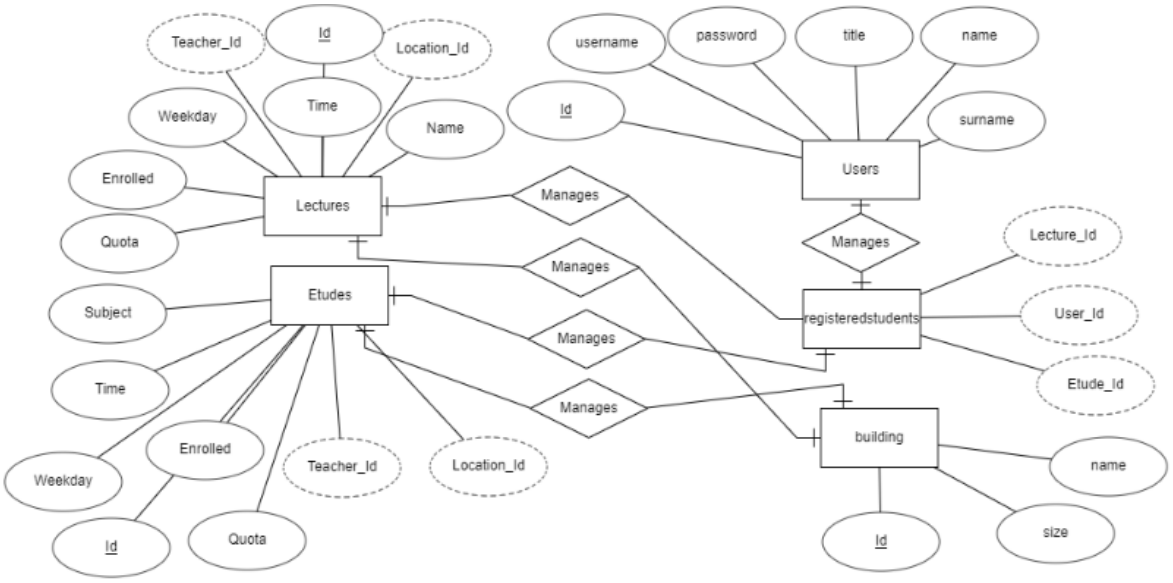

Fig. 2.1: E/R diagram for managers, teachers and students.

2.1.2 Code

The implementation of tables in the database is done using the following code block:

```
INIT_STATEMENTS = [
    """CREATE TABLE if not exists Users ( id SERIAL PRIMARY KEY,
username VARCHAR(255) NOT NULL UNIQUE,
password VARCHAR(255) NOT NULL,
title VARCHAR(255) NOT NULL,
```

```
name VARCHAR(255) NOT NULL,
surname VARCHAR(255) NOT NULL );""",

"""CREATE TABLE if not exists Buildings ( id SERIAL PRIMARY KEY,
name varchar(255) NOT NULL,
size int);""",

"""CREATE TABLE if not exists Lectures ( id SERIAL PRIMARY KEY,
name VARCHAR(255),
time TIME,
weekday VARCHAR(255),
enrolled int DEFAULT 0,
location_id int,
FOREIGN KEY (location_id)REFERENCES Buildings(id),
teacher_id int,
FOREIGN KEY (teacher_id) REFERENCES Teachers (id),
quota int,CONSTRAINT time_cons UNIQUE(time,weekday,location_id),
CONSTRAINT teacher_cons UNIQUE(time,weekday,teacher_id));""",

"""CREATE TABLE if not exists Etudes ( id SERIAL PRIMARY KEY,
subject VARCHAR(255),
teacher_id int,
FOREIGN KEY (teacher_id) REFERENCES Teachers(id),
time TIME,
weekday VARCHAR(255),
enrolled int DEFAULT 0,
location_id int NOT NULL,
FOREIGN KEY (location_id) REFERENCES Buildings(id),
quota int);""",

"""CREATE TABLE if not exists RegisteredStudents (
lecture_id int,
FOREIGN KEY (lecture_id) REFERENCES Lectures(id),
student_id int ,
FOREIGN KEY (student_id) REFERENCES Users(id),
etude_id int ,
FOREIGN KEY (etude_id) REFERENCES Etudes(id),
CONSTRAINT reg_cons UNIQUE(student_id,lecture_id),
CONSTRAINT regi_cons UNIQUE(student_id,etude_id));""",
]
```

2.1.3 Sign Up Function

The sign-up function implemented as a function when the form is posted from the HTML page. It first gets posted data from the form and if it is not in error, it inserts new users to the Users table. Then it also inserts other tables (Students, Teachers, Managers) according to given title information.

```
@app.route('/signup', methods=['POST'])
def my_form_post():
```

```
    username = request.form['UserName']
    password = pbkdf2_sha256.hash(request.form['Password'])
    firstname = request.form['FirstName']
    lastname = request.form['LastName']
    title = request.form.getlist('title')
    title = title[0]
    userid = 0
    statement = """INSERT INTO Users (username, password, title, name, surname)
    VALUES ('%s', '%s', '%s', '%s', '%s');""" % (username, password, title, firs
    statement += """select id from users where username = '%s'""" % (username)
    with psycopg2.connect(url) as connection:
        with connection.cursor() as cursor:
            cursor.execute(statement)
            userid += cursor.fetchone()[0]
    statement = """"""
    if title == "Student":
        degree = request.form['Degree']
        statement += """INSERT INTO Students (name,surname,degree,grade,user_id)
        VALUES ('%s', '%s', '%s', NULL,'%s');""" % (firstname, lastname, degree,
    if title == "Teacher":
        print(request.form)
        experience_year = request.form['Experience']
        subject = request.form['subject']
        statement += """INSERT INTO Teachers (name,surname,subject,experience_ye
        VALUES ('%s', '%s', '%s', '%s','%s');""" % (firstname, lastname, subject
    if title == "Manager":
        experience_year = request.form['Experience']
        print(experience_year)
        statement += """INSERT INTO Managers (name,surname,user_id,experience_ye
        VALUES ('%s', '%s', '%s', '%s');""" % (firstname, lastname, userid, expe
    with psycopg2.connect(url) as connection:
        with connection.cursor() as cursor:
            cursor.execute(statement)
            return render_template("homepage.html")
```

2.1.4 Lectures Function

Lecture function can be seen as 2 parts. These parts are according to the title of the user:

1. If the user is Teacher, HTML select tags prepared to use in lectures.html template. After the select tags prepared, it checks if a form is being posted. If a form is posted, it takes selected values from select tags in the form and it checks if the lecture information overlaps with any other lecture and if it overlaps an error message is sent to render_template. If no error occurs it runs SQL query to insert new lecture to the table.

2. If the user is a student, it first implements an HTML list for open lectures using a select SQL statement to get existing lectures and then it adds them to the template. After this task completed it checks again a form post. If form posted it detects which row of lecture selected using checked radio button. Then it gets the id of lecture to use the registry. Then it checks if the student has another lecture that overlaps with a new lecture if it is then an error message is sent to render_template. If no error occurs then it inserts lecture id to *registeredstudents* table

and user id also to use it in other processes. The function is then completed.

```
@login_required
@app.route("/lectures", methods=['GET', 'POST'])
def lectures():
    if not current_user.is_authenticated:
        return redirect("/")
    title = """"""
    statement = """select title from users where username = '%s'""" % (current_u
    with psycopg2.connect(url) as connection:
        with connection.cursor() as cursor:
            cursor.execute(statement)
            title = cursor.fetchone()[0]
    if title == "Teacher":
        error = """"""
        names = ["Physics", "Biology", "Chemistry"]
        days = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday"]
        times = ["9:30", "11:30", "13:30", "15:30"]
        locations = ["MED A34", "EHB 2101", "MDB A105"]
        quotas = ['1', '30', '45', '60']
        branches = """"""
        day = """"""
        time = """"""
        location = """"""
        quota = """"""
        for i in names:
            branches += """<option value="%s">%s</option>""" % (i, i)
        for i in days:
            day += """<option value="%s">%s</option>""" % (i, i)
        for i in times:
            time += """<option value="%s">%s</option>""" % (i, i)
        for i in locations:
            location += """<option value="%s">%s</option>""" % (i, i)
        for i in quotas:
            quota += """<option value="%s">%s</option>""" % (i, i)
        if request.method == "POST":
            branch = request.form.get("Branch", None)
            weekday = request.form.get("day", None)
            lecturetime = request.form.get("time", None)
            lecturelocation = request.form.get("location", None)
            lecturequota = request.form.get("quota", None)
            statement = """
            INSERT INTO Lectures (name, time, weekday, location_id, quota,teache
            VALUES('%s','%s','%s',(SELECT id from Buildings where name = '%s'),'
                branch, lecturetime, weekday, lecturelocation, lecturequota, cur
            with psycopg2.connect(url) as connection:
                with connection.cursor() as cursor:
                    try:
                        cursor.execute(statement)
                    except psycopg2.DatabaseError:
                        error = """You have already have a lecture at the same t
```

```
        return render_template("lectures.html", branchnames=branches, day=day, t
                               quota=quota, title=title, message=error)
    else:
        error = """"""
        if request.method == "POST":
            k = 0
            err = 0
            for i in range(11, 30000, 10):
                id = """%d""" % (i)
                subject = request.form.get(id, None)
                if subject is not None:
                    k = i - 1
                    break
            print(k)
            if k == 0:
                return redirect("/lectures")
            id = """%d""" % (k)
            lidd = request.form.get(id, None)
            if lidd is not None:
                statement1 = """insert into registeredstudents (lecture_id,stude
                lidd, current_user.username, lidd)
                statement2 = """select lecture_id from registeredstudents where
                    current_user.username)
                statement4 = """select weekday,time from lectures where id = %s"
                with psycopg2.connect(url) as connection:
                    with connection.cursor() as cursor:
                        cursor.execute(statement2)
                        reglec = cursor.fetchall()
                        cursor.execute(statement4)
                        lecc = cursor.fetchall()
                        lweekd = lecc[0][0]
                        ltimee = lecc[0][1]
                        if reglec is not None:
                            for row in reglec:
                                statement3 = """select weekday,time from lecture
                                cursor.execute(statement3)
                                wreglec = cursor.fetchall()
                                for lrow in wreglec:
                                    if lrow[0] == lweekd and lrow[1] == ltimee:
                                        error = """You have already have a lectu
                                        err = 1
                        if (err == 0):
                            cursor.execute(statement1)

        statement = """select id,name,weekday, time, quota from lectures"""
        lecturerows = """"""
        with psycopg2.connect(url) as connection:
            with connection.cursor() as cursor:
                cursor.execute(statement)
                i = 10
                for rows in cursor.fetchall():
```

```
                    lid = rows[0]
                    branch = rows[1]
                    weekday = rows[2]
                    lecturetime = rows[3]
                    lecturequota = rows[4]
                    lecturerows += """<tr><td>%s<input type="hidden" name="%d" v
                      <td><input id="%d" onclick="uncheck(%d)" type="radio" name
                    lid, i, lid,
                    branch, weekday, lecturetime, lecturequota, i + 1, i + 1, i
                    i += 10
        return render_template("lectures.html", title=title, newrow=lecturerows,
```

2.1.5 Etudes Function

Etudes function can be seen as 2 parts. These parts are according to the title of the user:

1. If the user is Teacher, HTML select tags prepared to use in the etudes.html template. After the select tags prepared, it checks if a form is being posted. If a form is a post, it takes selected values from select tags in the form and it checks if the etude information overlaps with any other etude and if it overlaps an error message is sent to render_template. If no error occurs it runs SQL query to insert new etude to the table.

2. If the user is a student, it first implements an HTML list for open etudes using a select SQL statement to get existing etudes and then it adds them to the template. After this task completed it checks again a form post. If form posted it detects which row of etude selected using checked radio button. Then it gets the id of etude to use the registry. Then it checks if the student has another etude that overlaps with a new etude if it is then an error message is sent to render_template. If no error occurs then it inserts etude id to *registeredstudents* table and user id also to use it in other processes. The function is then completed.

```
@login_required
@app.route("/etudes", methods=['GET', 'POST'])
def etudes():
    if not current_user.is_authenticated:
        return redirect("/")
    title = """"""
    statement = """select title from users where username = '%s'""" % (current_u
    with psycopg2.connect(url) as connection:
        with connection.cursor() as cursor:
            cursor.execute(statement)
            title = cursor.fetchone()[0]
    if title == "Teacher":
        with psycopg2.connect(url) as connection:
            with connection.cursor() as cursor:
                statement = """select subject from Teachers where user_id = (sel
                    current_user.username)
                cursor.execute(statement)
                subject = cursor.fetchone()[0]
        error = """"""
        if subject == "Physics":
            names = ["Electrics", "Acceleration", "Magnetism"]
```

```
        elif subject == "Biology":
            names = ["Bacteria", "Cell Division", "DNA"]
        elif subject == "Chemistry":
            names = ["Compounds", "Organics", "Acids"]
        days = ["Friday", "Saturday"]
        times = ["9:30", "11:30", "13:30", "15:30", "16:30", "17:30"]
        locations = ["MED A34", "EHB 2101", "MDB A105"]
        quotas = ['1', '30', '45', '60']
        branches = """"""
        day = """"""
        time = """"""
        location = """"""
        quota = """"""
        for i in names:
            branches += """<option value="%s">%s</option>""" % (i, i)
        for i in days:
            day += """<option value="%s">%s</option>""" % (i, i)
        for i in times:
            time += """<option id = "%s" value="%s">%s</option>""" % (i, i, i)
        for i in locations:
            location += """<option value="%s">%s</option>""" % (i, i)
        for i in quotas:
            quota += """<option value="%s">%s</option>""" % (i, i)
        if request.method == "POST":
            branch = request.form.get("Branch", None)
            weekday = request.form.get("day", None)
            lecturetime = request.form.get("time", None)
            lecturelocation = request.form.get("location", None)
            lecturequota = request.form.get("quota", None)
            statement = """
            INSERT INTO Etudes (subject, time, weekday, location_id, quota,teach
            VALUES('%s','%s','%s',(SELECT id from Buildings where name = '%s'),'
                branch, lecturetime, weekday, lecturelocation, lecturequota, cur
            with psycopg2.connect(url) as connection:
                with connection.cursor() as cursor:
                    try:
                        cursor.execute(statement)
                    except psycopg2.DatabaseError:
                        error = """You have already have a lecture at the same t
        return render_template("etudes.html", branchnames=branches, day=day, tim
                               quota=quota, title=title, message=error)
    else:
        error = """"""
        if request.method == "POST":
            k = 0
            err = 0
            for i in range(11, 30000, 10):
                id = """%d""" % (i)
                subject = request.form.get(id, None)
                if subject is not None:
                    k = i - 1
```

```
                break
            print(k)
            if k == 0:
                return redirect("/etudes")
            id = """%d""" % (k)
            lidd = request.form.get(id, None)
            statement1 = """insert into registeredstudents (etude_id,student_id)
            lidd, current_user.username, lidd)
            statement2 = """select etude_id from registeredstudents where studen
                current_user.username)
            statement4 = """select weekday,time from etudes where id = %s""" % (
            with psycopg2.connect(url) as connection:
                with connection.cursor() as cursor:
                    cursor.execute(statement2)
                    reglec = cursor.fetchall()
                    cursor.execute(statement4)
                    lecc = cursor.fetchall()
                    lweekd = lecc[0][0]
                    ltimee = lecc[0][1]
                    print(reglec)
                    if reglec is not None:
                        for row in reglec:
                            statement3 = """select weekday,time from etudes wher
                            cursor.execute(statement3)
                            wreglec = cursor.fetchall()
                            for lrow in wreglec:
                                if lrow[0] == lweekd and lrow[1] == ltimee:
                                    error = """You have already have a etude at
                                    err = 1
                    if (err == 0):
                        cursor.execute(statement1)

        statement = """select id,subject,weekday, time, quota from etudes"""
        lecturerows = """"""
        with psycopg2.connect(url) as connection:
            with connection.cursor() as cursor:
                cursor.execute(statement)
                i = 10
                for rows in cursor.fetchall():
                    lid = rows[0]
                    branch = rows[1]
                    weekday = rows[2]
                    lecturetime = rows[3]
                   lecturequota = rows[4]
                    lecturerows += """<tr><td>%s<input type="hidden" name="%d" v
                      <td><input id="%d" onclick="uncheck(%d)" type="radio" name
                    lid, i, lid,
                    branch, weekday, lecturetime, lecturequota, i + 1, i + 1, i
                    i += 10
        return render_template("etudes.html", title=title, newrow=lecturerows, m
```

2.1.6 Schedule Function

The schedule function is used to implement a schedule for teachers and students, and also to achieve lecture delete or unroll and for teachers updating lectures whenever they want.

Function parted into 2, first is to teachers and second to students. In the first part, it first checks if any delete, unroll or update request is sent from the form. If it then it first performs operations and updates tables in the database according to these operations. Then implement the HTML part of the schedule table using select statements.

In the second part, the user is a student. So an unroll operation should be checked and it checks first it anyone requested. Then it implements the HTML part using select statements, selects lectures and etudes from tables according to etude_id and lecture_ids from table *registeredstudents*.

```
@login_required
@app.route("/schedule", methods=['GET', 'POST'])
def schedule():
    if not current_user.is_authenticated:
        return redirect("/")
    statement = """SELECT title FROM Users WHERE username = '%s'""" % (current_u
    schedulerows = """<tr>"""
    with psycopg2.connect(url) as connection:
        with connection.cursor() as cursor:
            cursor.execute(statement)
            title = cursor.fetchone()[0]
            if title == "Manager":
                message = "Managers don't have a schedule."
                return render_template("homepage.html", message=message)
            others = title
            others += "s"
            if title == "Teacher":
                if request.method == "POST":
                    if request.form['sub'] == "Delete":
                        lecidd = request.form.get("LecID", None)
                        etdidd = request.form.get("EtudeID", None)
                        if lecidd is not None and lecidd != "":
                            statement = """delete from registeredstudents where
                            cursor.execute(statement)
                            statement = """delete from lectures where id = %s"""
                            cursor.execute(statement)
                        elif etdidd is not None and etdidd != "":
                            statement = """delete from registeredstudents where
                            cursor.execute(statement)
                            statement = """delete from etudes where id = %s""" %
                            cursor.execute(statement)
                    elif request.form['sub'] == "Update":
                        lecidd = request.form.get("LecID2", None)
                        etdidd = request.form.get("EtudeID2", None)
                        if lecidd is not None and lecidd != "":
                            statement = """update lectures set weekday = '%s', t
                                        where id = %s and teacher_id = (select id
```

```
                        request.form['wday'], request.form['timee'],
                        request.form['lquotaa'], request.form['lcation'], le
                        cursor.execute(statement)
                    elif etdidd is not None and etdidd != "":
                        statement = """update etudes set weekday = '%s', tim
                                    where id = %s and teacher_id = (select id
                        request.form['etudewday'], request.form['timee2'],
                        request.form['lquotaa'], request.form['lcation'], et
                        print(statement)
                        cursor.execute(statement)
            statement = """SELECT id,name, time, weekday, location_id, quota
        SELECT id FROM %s WHERE user_id = (SELECT id FROM Users WHERE userna
                        % (others, current_user.username)
            cursor.execute(statement)
            lname = ""
            lcrn = 0
            ltime = '00:00'
            lweekday = ""
            llocation_id = 0
            lquota = 0
            schedule1 = cursor.fetchall()
            statement = """SELECT id,subject, time, weekday, location_id, qu
        SELECT id FROM %s WHERE user_id = (SELECT id FROM Users WHERE userna
                        % (others, current_user.username)
            cursor.execute(statement)
            etdschedule = cursor.fetchall()
            if schedule1 is not None:
                Monday = """<th>"""
                Tuesday = """<th>"""
                Wednesday = """<th>"""
                Thursday = """<th>"""
                Friday = """<th>"""
                Saturday = """<th>"""
                for row in schedule1:
                    print(row)
                    lname = row[1]
                    lcrn = row[0]
                    ltime = row[2]
                    lweekday = row[3]
                    llocation_id = row[4]
                    lquota = row[5]
                    statement = """SELECT name FROM Buildings WHERE id = %s"
                    cursor.execute(statement)
                    llocation = ""
                    for row in cursor.fetchall():
                        print(row)
                        llocation = row[0]
                    if lweekday == "Monday":
                        Monday += """<p>Lecture ID: %s</p><p>%s</p><p>%s</p>
                        lcrn, lname, ltime, llocation, lquota)
                    elif lweekday == "Tuesday":
```

```
                        Tuesday += """<p>Lecture ID: %s</p><p>%s</p><p>%s</p
                        lcrn, lname, ltime, llocation, lquota)
                    elif lweekday == "Wednesday":
                        Wednesday += """<p>Lecture ID: %s</p><p>%s</p><p>%s<
                        lcrn, lname, ltime, llocation, lquota)
                    elif lweekday == "Thursday":
                        Thursday += """<p>Lecture ID: %s</p><p>%s</p><p>%s</
                        lcrn, lname, ltime, llocation, lquota)
                    elif lweekday == "Friday":
                        Friday += """<p>Lecture ID: %s</p><p>%s</p><p>%s</p>
                        lcrn, lname, ltime, llocation, lquota)
                if etdschedule is not None:
                    for row in etdschedule:
                        lname = row[1]
                        lcrn = row[0]
                        ltime = row[2]
                        lweekday = row[3]
                        llocation_id = row[4]
                        lquota = row[5]
                        statement = """SELECT name FROM Buildings WHERE id =
                        cursor.execute(statement)
                        llocation = ""
                        for row in cursor.fetchall():
                            print(row)
                            llocation = row[0]
                        if lweekday == "Friday":
                            Friday += """<p>Etude ID: %s</p><p>%s</p><p>%s</
                            lcrn, lname, ltime, llocation, lquota)
                        elif lweekday == "Saturday":
                            Saturday += """<p>Etude ID: %s</p><p>%s</p><p>%s
                            lcrn, lname, ltime, llocation, lquota)
                Monday += """</th>"""
                Tuesday += """</th>"""
                Wednesday += """</th>"""
                Thursday += """</th>"""
                Friday += """</th>"""
                Saturday += """</th>"""
                schedulerows += Monday + Tuesday + Wednesday + Thursday + Fr
                schedulerows += """</tr>"""
                return render_template("schedule.html", newrow=schedulerows,
            elif title == "Student":
                statement = """select lecture_id from registeredstudents where s
                            % (current_user.username)

                Monday = """<th>"""
                Tuesday = """<th>"""
                Wednesday = """<th>"""
                Thursday = """<th>"""
                Friday = """<th>"""
                Saturday = """<th>"""
                with psycopg2.connect(url) as connection:
```

```
                with connection.cursor() as cursor:
                    if request.method == "POST":
                        lecidd = request.form.get("LecID", None)
                        etdidd = request.form.get("EtudeID", None)
                        if lecidd is not None and lecidd != "":
                            statement1 = """delete from registeredstudents w
                             lectures set enrolled = enrolled - 1 where id =
                            lecidd, current_user.username, lecidd)
                            cursor.execute(statement1)
                        elif etdidd is not None and etdidd != "":
                            statement1 = """delete from registeredstudents w
                             etudes set enrolled = enrolled - 1 where id = %
                            etdidd, current_user.username, etdidd)
                            cursor.execute(statement1)
                        cursor.execute(statement)
                        lids = cursor.fetchall()
                        if lids is not None:
                            for lirow in lids:
                                lid = lirow[0]
                                if lid is not None:
                                    statement = """SELECT id,name, time, weekday
                                        lid)
                                    cursor.execute(statement)
                                    lname = ""
                                    lcrn = 0
                                    ltime = '00:00'
                                    lweekday = ""
                                    llocation_id = 0
                                    lquota = 0
                                    schedule1 = cursor.fetchall()
                                   if schedule1 is not None:
                                        for row in schedule1:
                                            print(row)
                                            lname = row[1]
                                            lcrn = row[0]
                                            ltime = row[2]
                                            lweekday = row[3]
                                            llocation_id = row[4]
                                            lquota = row[5]
                                            statement = """SELECT name FROM Buil
                                            cursor.execute(statement)
                                            llocation = ""
                                            for lorow in cursor.fetchall():
                                                print(lorow)
                                                llocation = lorow[0]
                                            if lweekday == "Monday":
                                                Monday += """<p>Lecture ID: %s</
                                                lcrn, lname, ltime, llocation, l
                                            elif lweekday == "Tuesday":
                                                Tuesday += """<p>Lecture ID: %s<
                                                lcrn, lname, ltime, llocation, l
```

```
                                    elif lweekday == "Wednesday":
                                        Wednesday += """<p>Lecture ID: %
                                        lcrn, lname, ltime, llocation, l
                                    elif lweekday == "Thursday":
                                        Thursday += """<p>Lecture ID: %s
                                        lcrn, lname, ltime, llocation, l
                                    elif lweekday == "Friday":
                                        Friday += """<p>Lecture ID: %s</
                                        lcrn, lname, ltime, llocation, l
                statement = """select etude_id from registeredstudents w
                            % (current_user.username)
                cursor.execute(statement)
                etdlids = cursor.fetchall()
                if etdlids is not None:
                    for etdrow in etdlids:
                        etdlid = etdrow[0]
                        if etdlid is not None:
                            statement = """SELECT id,subject, time, week
                                etdlid)
                            cursor.execute(statement)
                            etdschedule = cursor.fetchall()
                            lname = ""
                            lcrn = 0
                            ltime = '00:00'
                            lweekday = ""
                            llocation_id = 0
                            lquota = 0
                            if etdschedule is not None:
                                for row in etdschedule:
                                    lname = row[1]
                                    lcrn = row[0]
                                    ltime = row[2]
                                    lweekday = row[3]
                                    llocation_id = row[4]
                                    lquota = row[5]
                                    statement = """SELECT name FROM Buil
                                    cursor.execute(statement)
                                    llocation = ""
                                    for row in cursor.fetchall():
                                        print(row)
                                        llocation = row[0]
                                    if lweekday == "Friday":
                                        Friday += """<p>Etude ID: %s</p>
                                        lcrn, lname, ltime, llocation, l
                                    elif lweekday == "Saturday":
                                        Saturday += """<p>Etude ID: %s</
                                        lcrn, lname, ltime, llocation, l
                            Monday += """</th>"""
                            Tuesday += """</th>"""
                            Wednesday += """</th>"""
                            Thursday += """</th>"""
```

```
                                    Friday += """</th>"""
                                    Saturday += """</th>"""
                                    schedulerows += Monday + Tuesday + Wednesday
                                    print(schedulerows)
                                    schedulerows += """</tr>"""
                return render_template("schedule.html", newrow=schedulerows)
```

2.2 Parts Implemented by Burak Bozdağ

2.2.1 Database Design

I implemented three tables in this project: *Managers*, *Teachers* and *Students*.

Managers manages teachers and students. Teachers teaches students. Teachers gives lectures and etudes to students. Students takes lectures and etudes from teachers. Managers also manages these lectures and etudes. The E/R diagram is shown below:

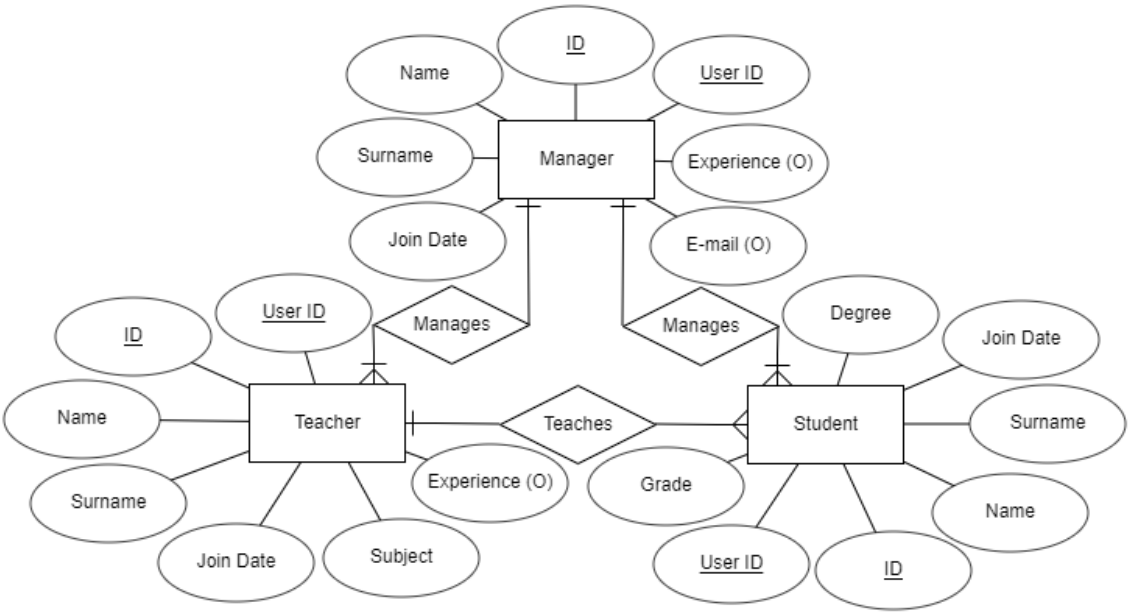

Fig. 2.2: E/R diagram for managers, teachers and students.

2.2.2 Code

The implementation of tables in the database is done using the following code block:

```
INIT_STATEMENTS = [
    """CREATE TABLE if not exists Managers ( id SERIAL PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    surname VARCHAR(255) NOT NULL,
    email VARCHAR(255),
    join_date DATE NOT NULL DEFAULT CURRENT_DATE,
```

```
    experience_year int,
    user_id int NOT NULL,
    FOREIGN KEY (user_id) REFERENCES Users(id));""",

    """CREATE TABLE if not exists Teachers ( id SERIAL PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    surname VARCHAR(255) NOT NULL,
    subject varchar(255),
    join_date DATE NOT NULL DEFAULT CURRENT_DATE,
    experience_year int,
    user_id int NOT NULL,
    FOREIGN KEY (user_id) REFERENCES Users(id));""",

    """CREATE TABLE if not exists Students ( id SERIAL PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    surname VARCHAR(255) NOT NULL,
    degree int NOT NULL,
    join_date DATE NOT NULL DEFAULT CURRENT_DATE,
    grade FLOAT,
    user_id int NOT NULL,
    FOREIGN KEY (user_id) REFERENCES Users(id));""",
]
```

Database Functions

The most used database functions are implemented in a separate file called *database.py* to avoid redundant codes in *server.py*.

Getting user

```
def get_user(self, username):
    user = object()
    user.username = username
    with dbapi2.connect(self.dbfile) as connection:
        cursor = connection.cursor()
        query = """SELECT username, password, title, name, surname FROM Users WH
        cursor.execute(query)
        curs = cursor.fetchall()
        user.password = curs[1]
        user.title = curs[2]
        user.name = curs[3]
        user.surname = curs[4]
    return user
```

Updating user

```
def update_user(self, user_id, user):
    with dbapi2.connect(self.dbfile) as connection:
        cursor = connection.cursor()
        query = """UPDATE Users
```

```
        SET " username = '%s', password = '%s' title = '%s' name = '%s' surname
            user.username, user.password, user.title, user.name, user.surname, u
        cursor.execute(query)
        connection.commit()
```

Updating Profile

```
def update_profile(self, user_id, profile):  # Changeable attributes: name, surn
    with dbapi2.connect(self.dbfile) as connection:
        cursor = connection.cursor()
        query = "SELECT title FROM Users WHERE id = %s" % (user_id,)
        cursor.execute(query)
        title = cursor.fetchone()[0]
        title += "s"
        query = "UPDATE %s SET name = '%s', YR = %s WHERE (ID = %s)" % (
            title, profile.name, profile.surname, profile.id)
        cursor.execute(query)
        connection.commit()
```

Deleting User

```
def delete_user(self, user_id, profile_id):  # Deletes user and profile
    with dbapi2.connect(self.dbfile) as connection:
        cursor = connection.cursor()
        query = "SELECT title FROM Users WHERE id = %s" % (user_id,)
        cursor.execute(query)
        title = cursor.fetchone()[0]
        title += "s"
        query = """DELETE FROM %s WHERE (id = %s)""" % (title, profile_id)
        cursor.execute(query)
        query = """DELETE FROM Users WHERE (id = %s)""" % (user_id,)
        cursor.execute(query)
        connection.commit()
```

There are many view functions such as login, home page, profile, etc. These functions are described in next parts.

Log In and Log Out

When logging in, the server checks whether the username exists in the database. Then, password checking is done using hashing methods. If these requirements are satisfied, the user logs into the site successfully.

The following code block is responsible for logging in:

```
@app.route("/login", methods=['POST'])
def login():
    username = request.form['UserName']
    user = get_user(username)
    if user is not None:
```

```
        password = request.form['Password']
        password1 = user.password
        if pbkdf2_sha256.verify(password, password1):
            login_user(user)
            flash("You have logged in.")
            return redirect("/")
    return render_template("signin.html", message="Invalid credentials.")
```

When logging out, the server checks whether a user is already logged in, then proceeds to the logging out process.

The following code block is responsible for logging out:

```
@login_required
@app.route("/logout")
def logout():
    if not current_user.is_authenticated:
        return redirect("/")
    logout_user()
    return render_template("homepage.html", message="You have logged out.")
```

The Profile Page

View Profile Attributes

If a user wants to look at his/her information that is stored in the database, the user can enter to the profile page. The profile view function displays all attributes that a user has in the output.

The following code block is responsible for displaying profile attributes:

```
@login_required
@app.route("/profile")
def profile():
    if not current_user.is_authenticated:
        return redirect("/")
    titles = """select title from users where username = '%s'""" % (current_user
    with psycopg2.connect(url) as connection:
        with connection.cursor() as cursor:
            cursor.execute(titles)
            title = cursor.fetchone()[0]
            if title == "Manager":
                statement = """SELECT * FROM Managers WHERE user_id = (SELECT id
                            % (current_user.username,)
                cursor.execute(statement)
                for row in cursor.fetchall():
                    print(row)
                    name = row[1]
                    surname = row[2]
                    email = row[3]
                    join_date = row[4]
                    experience = row[5]
                return render_template("profile.html",
```

```
                                    username=current_user.username, title=tit
                                    email=email, join_date=join_date, experie
            elif title == "Teacher":
                statement = """SELECT * FROM Teachers WHERE user_id = (SELECT id
                            % (current_user.username,)
                cursor.execute(statement)
                for row in cursor.fetchall():
                    print(row)
                    name = row[1]
                    surname = row[2]
                    subject = row[3]
                    join_date = row[4]
                    experience = row[5]
                return render_template("profile.html",
                                    username=current_user.username, title=tit
                                    subject=subject, join_date=join_date, exp
            elif title == "Student":
                statement = """SELECT * FROM Students WHERE user_id = (SELECT id
                            % (current_user.username,)
                cursor.execute(statement)
                for row in cursor.fetchall():
                    print(row)
                    name = row[1]
                    surname = row[2]
                    degree = row[3]
                    join_date = row[4]
                    grade = row[5]
                return render_template("profile.html",
                                    username=current_user.username, title=tit
                                    degree=degree, join_date=join_date, grade
            else:
                abort(404)
```

Update Profile Attributes

The user may want to update his/her attributes. For this problem, there is a function that lets the user to update information in the database.

The following code block is responsible for updating profile:

```
@login_required
@app.route("/update-profile")
def profile_update():
    if not current_user.is_authenticated:
        return redirect("/")
    query = """SELECT id, title FROM Users WHERE username = '%s'""" % (current_u
    with psycopg2.connect(url) as connection:
        with connection.cursor() as cursor:
            cursor.execute(query)
            for row in cursor.fetchall():
                print(row)
```

```
                user_id = row[0]
                title = row[1]
            if title == "Manager":
                query = """SELECT name, surname, email, experience_year FROM Man
                cursor.execute(query)
                for row in cursor.fetchall():
                    name = row[0]
                    surname = row[1]
                    email = row[2]
                    experience = row[3]
                return render_template("update-profile.html", name=name, surname
                                       experience=experience)
            elif title == "Teacher":
                query = """SELECT name, surname, subject, experience_year FROM T
                cursor.execute(query)
                for row in cursor.fetchall():
                    name = row[0]
                    surname = row[1]
                    subject = row[2]
                    experience = row[3]
                return render_template("update-profile.html",
                                       name=name, surname=surname, subject=subje
            elif title == "Student":
                query = """SELECT name, surname, degree FROM Students WHERE user
                cursor.execute(query)
                for row in cursor.fetchall():
                    name = row[0]
                    surname = row[1]
                    degree = row[2]
                return render_template("update-profile.html", name=name, surname
            else:
                abort(404)
```

In order to submit changes to update the profile, there must be a route for *POST* method to update the database:

```
@login_required
@app.route("/update-profile", methods=['POST'])
def update_profile():
    if not current_user.is_authenticated:
        return redirect("/")
    query = """SELECT id, title FROM Users WHERE username = '%s'""" % (current_u
    with psycopg2.connect(url) as connection:
        with connection.cursor() as cursor:
            cursor.execute(query)
            for row in cursor.fetchall():
                print(row)
                user_id = row[0]
                title = row[1]
            if title == "Manager":
                name = request.form['name']
```

```
                surname = request.form['surname']
                email = request.form['email']
                experience = request.form['experience']
                query = """UPDATE Managers SET name = '%s', surname = '%s', emai
                WHERE user_id = %s""" % (name, surname, email, experience, user_
                cursor.execute(query)
                query = """UPDATE Users SET name = '%s', surname = '%s' WHERE id
                cursor.execute(query)
                connection.commit()
            elif title == "Teacher":
                name = request.form['name']
                surname = request.form['surname']
                subject = request.form['subject']
                experience = request.form['experience']
                query = """UPDATE Teachers SET name = '%s', surname = '%s', subj
                WHERE user_id = %s""" % (name, surname, subject, experience, use
                cursor.execute(query)
                query = """UPDATE Users SET name = '%s', surname = '%s' WHERE id
                cursor.execute(query)
                connection.commit()
            elif title == "Student":
                name = request.form['name']
                surname = request.form['surname']
                degree = request.form['degree']
                query = """UPDATE Students SET name = '%s', surname = '%s', degr
                    name, surname, degree, user_id)
                cursor.execute(query)
                query = """UPDATE Users SET name = '%s', surname = '%s' WHERE id
                cursor.execute(query)
                connection.commit()
```

Updating Password

If the user wants to update the account's password, it is done by entering the old password and the new password. The password function checks whether the old password is correct, then checks whether two new passwords that are entered are matching. At last, the server updates the account's password.

The following code block is responsible for updating password:

```
@login_required
@app.route("/password", methods=['POST'])
def change_password():
    if not current_user.is_authenticated:
        return redirect("/")
    new1 = request.form['new1']
    new2 = request.form['new2']
    if new1 != new2:
        message = "New passwords don't match. Try again."
        return render_template("password.html", message=message)
    old = request.form['old']
```

```
    user = get_user(current_user.username)
    if pbkdf2_sha256.verify(old, user.password):
        new = pbkdf2_sha256.hash(new1)
        query = """UPDATE Users SET password = '%s' WHERE username = '%s'""" % (
        with psycopg2.connect(url) as connection:
            with connection.cursor() as cursor:
                cursor.execute(query)
                connection.commit()
                logout_user()
                return redirect("/signin")
    else:
        message = "The old password you entered is incorrect. Try again."
        return render_template("password.html", message=message)
```

Deleting Profile

The user might want to delete his/her account from the database but this is a problematic wish since there are many relations between managers, teachers and students. The server checks whether the user is dropped from all lectures, etudes, etc. then allows the profile deleting process.

The following code block is responsible for deleting a profile:

```
@login_required
@app.route("/DELETE")
def delete():
    if not current_user.is_authenticated:
        return redirect("/")
    query = """SELECT id, title FROM Users WHERE username = '%s'""" % (current_u
    with psycopg2.connect(url) as connection:
        with connection.cursor() as cursor:
            cursor.execute(query)
            for row in cursor.fetchall():
                user_id = row[0]
                title = row[1]
            title += "s"
            query = """DELETE FROM %s WHERE user_id = %s""" % (title, user_id)
            try:
                cursor.execute(query)
            except psycopg2.DatabaseError:
                if title == "Teachers":
                    message = "A teacher should quit from his/her lectures/etude
                    return render_template("homepage.html", message=message)
                else:
                    abort(404)
            logout_user()
            if title == "Students":
                query = """DELETE FROM RegisteredStudents WHERE student_id = %s"
                cursor.execute(query)
            query = """DELETE FROM Users WHERE id = %s""" % user_id
            cursor.execute(query)
```

```
            connection.commit()
            return redirect("/")
```

Authorizations for Specific Users

In *SchoolManager*, there are some actions that are used by managers and teachers. In the following sections, these authorizations are explained.

Viewing All Students

Only managers and teachers can view all students that are in the database. This function only works when the current user is not a student.

The following code block is responsible for displaying all students that are in the database:

```
@login_required
@app.route("/students")
def students():
    if not current_user.is_authenticated:
        return redirect("/")
    query = """SELECT title FROM Users WHERE username = '%s'""" % (current_user.
    with psycopg2.connect(url) as connection:
        with connection.cursor() as cursor:
            cursor.execute(query)
            title = cursor.fetchone()[0]
            if title != "Student":
                query = """SELECT * from Students ORDER BY id"""
                cursor.execute(query)
                rows = []
                for row in cursor.fetchall():
                    print(row)
                    rows += (row,)
                print(rows)
                return render_template("students.html", students=rows)
            else:
                return redirect("/")
```

Grading

Managers and teachers are able to grade their students using the previous function that displays all students. The user enter a student ID inside a form to update a student's grade:

```
@login_required
@app.route("/student", methods=['POST'])
def give_grade():
    if not current_user.is_authenticated:
        return redirect("/")
    student_id = request.form['id']
    query = """SELECT name, surname, degree, join_date, grade FROM Students WHER
    with psycopg2.connect(url) as connection:
        with connection.cursor() as cursor:
```

```
            cursor.execute(query)
            for row in cursor.fetchall():
                if row[0] is None:
                    return redirect("/students")
                elif row[0] == "":
                    return redirect("/students")
                name = row[0]
                surname = row[1]
                degree = row[2]
                join_date = row[3]
                grade = row[4]
                return render_template("student.html", id=student_id, name=name,
                                       join_date=join_date, grade=grade)
            return redirect("/students")
```

When the new grade is entered, the server updates the grade of the student using the following code block:

```
@login_required
@app.route("/grade", methods=['POST'])
def grader():
    if not current_user.is_authenticated:
        return redirect("/")
    query = """UPDATE Students SET grade = %s WHERE id = %s""" % (request.form['
    with psycopg2.connect(url) as connection:
        with connection.cursor() as cursor:
            cursor.execute(query)
            connection.commit()
            return redirect("/students")
```

Viewing All Teachers

This function displays all teachers that are in the database. Only managers can use this function. This limitation is checked by getting the current user's title.

The following code is responsible for displaying all teachers that are in the database:

```
@login_required
@app.route("/teachers")
def teachers():
    if not current_user.is_authenticated:
        return redirect("/")
    query = """SELECT title FROM Users WHERE username = '%s'""" % (current_user.
    with psycopg2.connect(url) as connection:
        with connection.cursor() as cursor:
            cursor.execute(query)
            title = cursor.fetchone()[0]
            if title == "Manager":
                query = """SELECT * from Teachers ORDER BY id"""
                cursor.execute(query)
                rows = []
```

```
                for row in cursor.fetchall():
                    print(row)
                    rows += (row,)
                print(rows)
                return render_template("teachers.html", teachers=rows)
            else:
                return redirect("/")
```