Marcin Miszkiel (432418)

Katarzyna Mocio (429956)

Final Project Report: *Electronic Journal*

SHORT INTRODUCTION

Electronic Journal is a web app written in Python in the Flask framework for web applications. The application is similar to the existing on the market electronic school journals in a simplified version, with the possibility of further expansion. The database was created in SQLite and consists of tables connected with multiple references. The application uses complex SQL queries. Safeguards have been put in place to protect user data.

The app allows users to perform actions typical of school electronic journals. Teachers can enter new grades into the system and view the grades they have already entered for the subjects they teach in the classes they teach, as well as various useful statistics related to grades. The panel available to students also allows for a broad view of the grades they have received and additionally provides information in comparison to the class to which they are assigned. Admins enter new users or subject to the system and assign a class to students, and a class and subject to teachers.

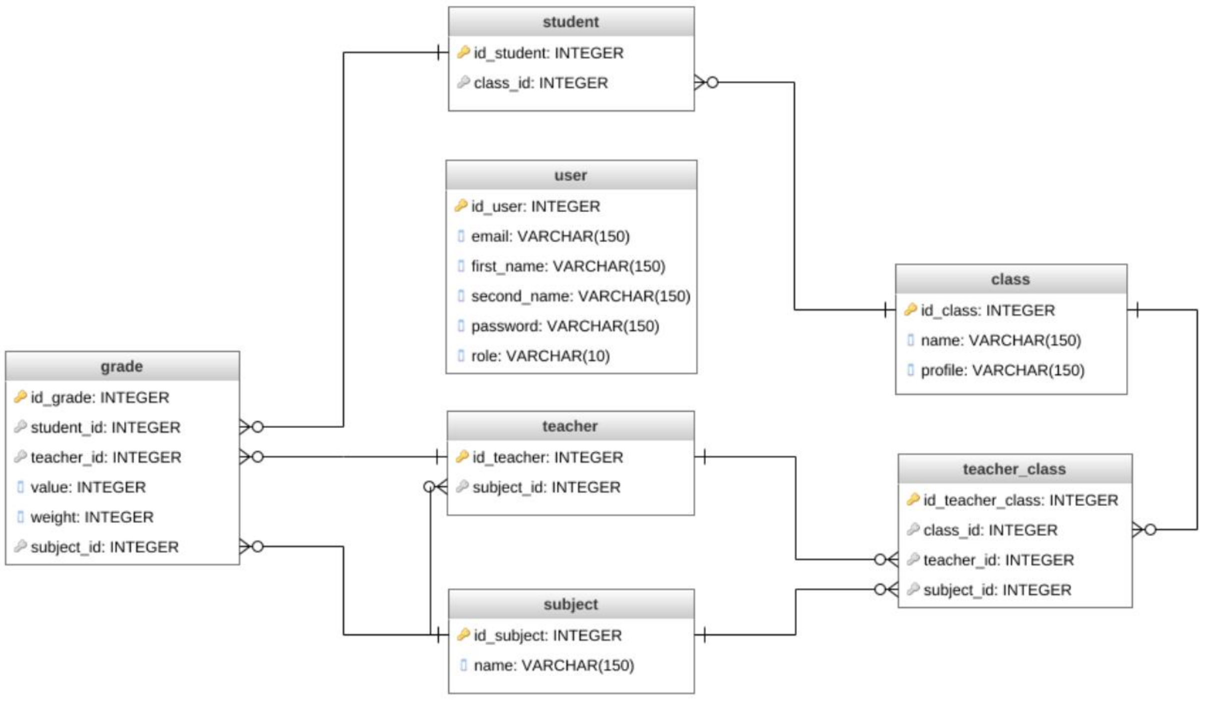
Warsaw, January 2024

DATABASE

Our comprehensive database, composed of seven interconnected tables, establishes intricate connections among students, classes, teachers, subjects, and grades through multiple references. These tables collectively form a dataset, capturing vital information that mirrors the dynamic interactions within the educational framework. Specifically, it ensures a comprehensive representation of student-class affiliations, teacher-subject associations, and the evaluation of students' academic performance. This methodical approach provides a sturdy foundation for the management and analysis of data in an school grades context.

Moreover, the 'role' column in the 'user' table plays a pivotal role, distinguishing between admin, teacher, or student roles. This role-based differentiation is utilized to establish inheritance relationships between the 'teacher' and 'student' tables with the 'user' table. In practice, both the 'teacher' and 'student' tables reference the 'user' table, inheriting pertinent information based on the user's role. This strategic design decision streamlines the representation of administrators, teachers, and students within the database, enhancing the overall data structure and clarity.

Pic. 1. Database scheme

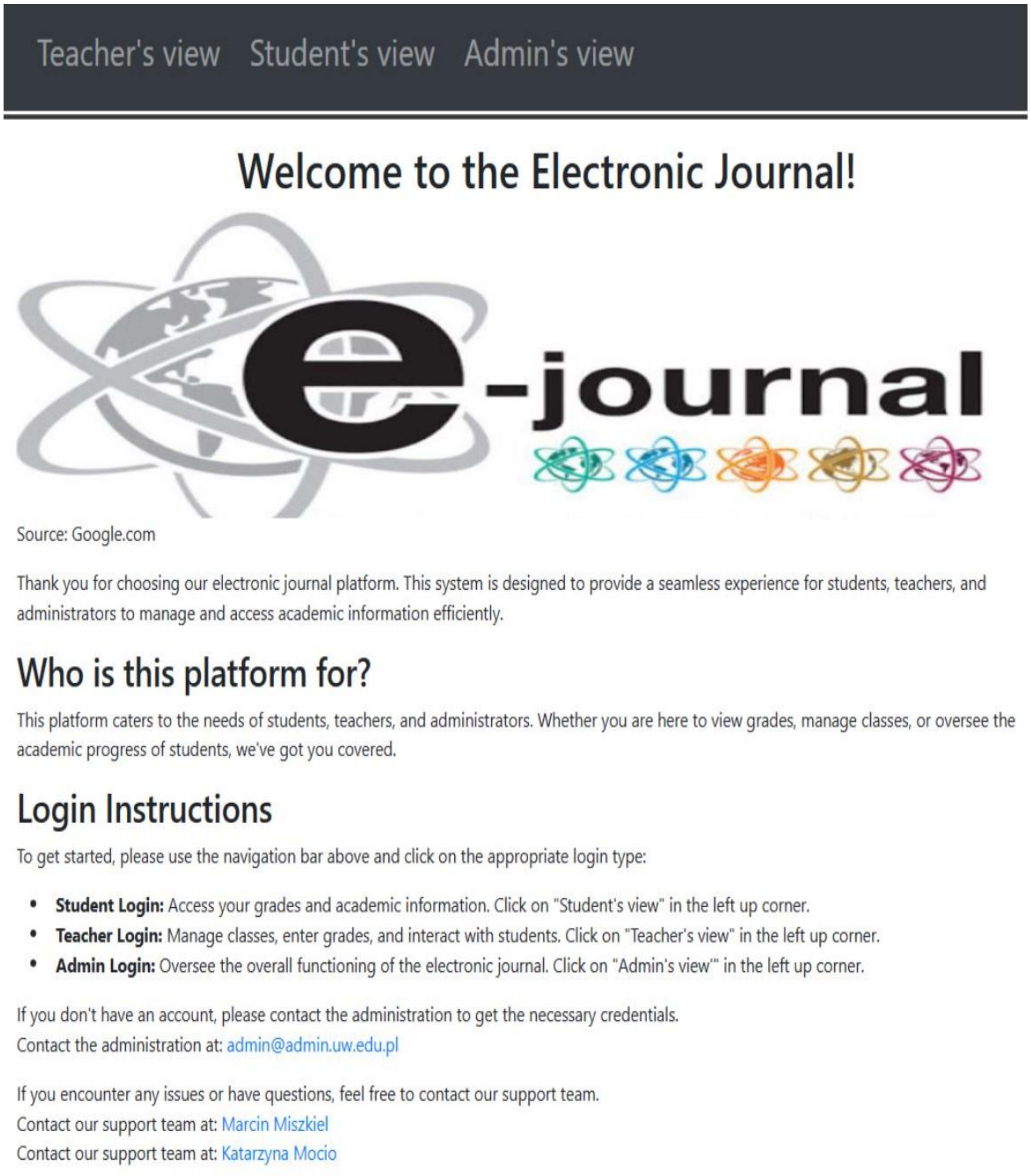


Source: Own work, based on: app.genmymodel.

USER AUTHENTICATION

User authentication is a fundamental aspect of the application, ensuring secure access to distinct dashboards based on the user's role. The system supports roles such as admin, teacher, and student. Upon successful authentication, users are redirected to their respective dashboards. This robust authentication mechanism enhances the overall security and usability of the application. But before login first page which user can see it is a welcome page:

Pic. 2. Welcome Page



Welcome to the Electronic Journal!

Source: Google.com

Thank you for choosing our electronic journal platform. This system is designed to provide a seamless experience for students, teachers, and administrators to manage and access academic information efficiently.

Who is this platform for?

This platform caters to the needs of students, teachers, and administrators. Whether you are here to view grades, manage classes, or oversee the academic progress of students, we've got you covered.

Login Instructions

To get started, please use the navigation bar above and click on the appropriate login type:

- **Student Login:** Access your grades and academic information. Click on "Student's view" in the left up corner.
- **Teacher Login:** Manage classes, enter grades, and interact with students. Click on "Teacher's view" in the left up corner.
- **Admin Login:** Oversee the overall functioning of the electronic journal. Click on "Admin's view'" in the left up corner.

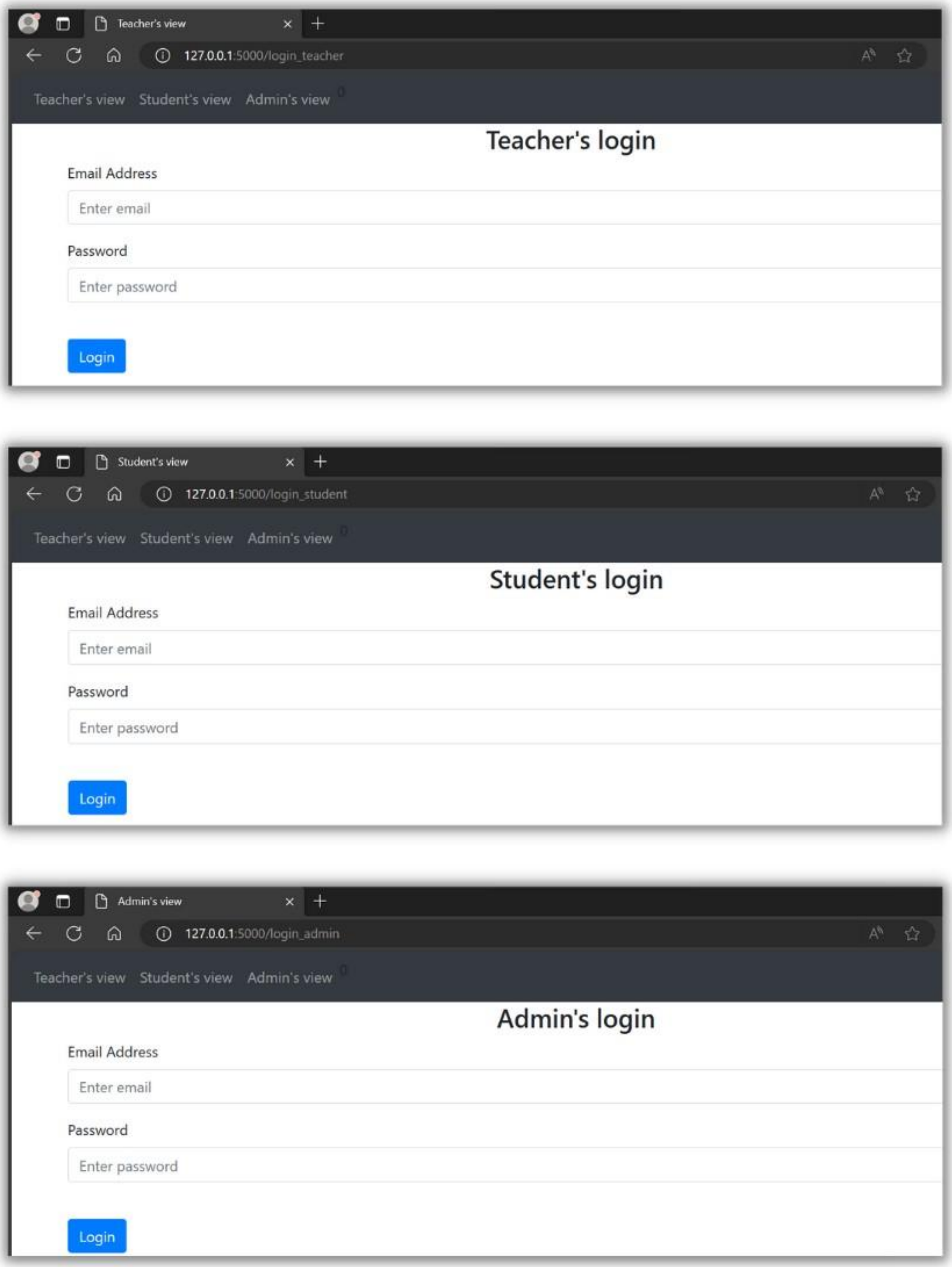
If you don't have an account, please contact the administration to get the necessary credentials.
Contact the administration at: admin@admin.uw.edu.pl

If you encounter any issues or have questions, feel free to contact our support team.
Contact our support team at: Marcin Miszkiel
Contact our support team at: Katarzyna Mocio

Source: Own work, based on Python, flask.

And the login panels:

Pic. 3. Login Pages


Teacher's view
127.0.0.1:5000/login_teacher
Teacher's view Student's view Admin's view

Teacher's login

Email Address

Enter email

Password

Enter password

Login
Student's view
127.0.0.1:5000/login_student
Teacher's view Student's view Admin's view

Student's login

Email Address

Enter email

Password

Enter password

Login
Admin's view
127.0.0.1:5000/login_admin
Teacher's view Student's view Admin's view

Admin's login

Email Address

Enter email

Password

Enter password

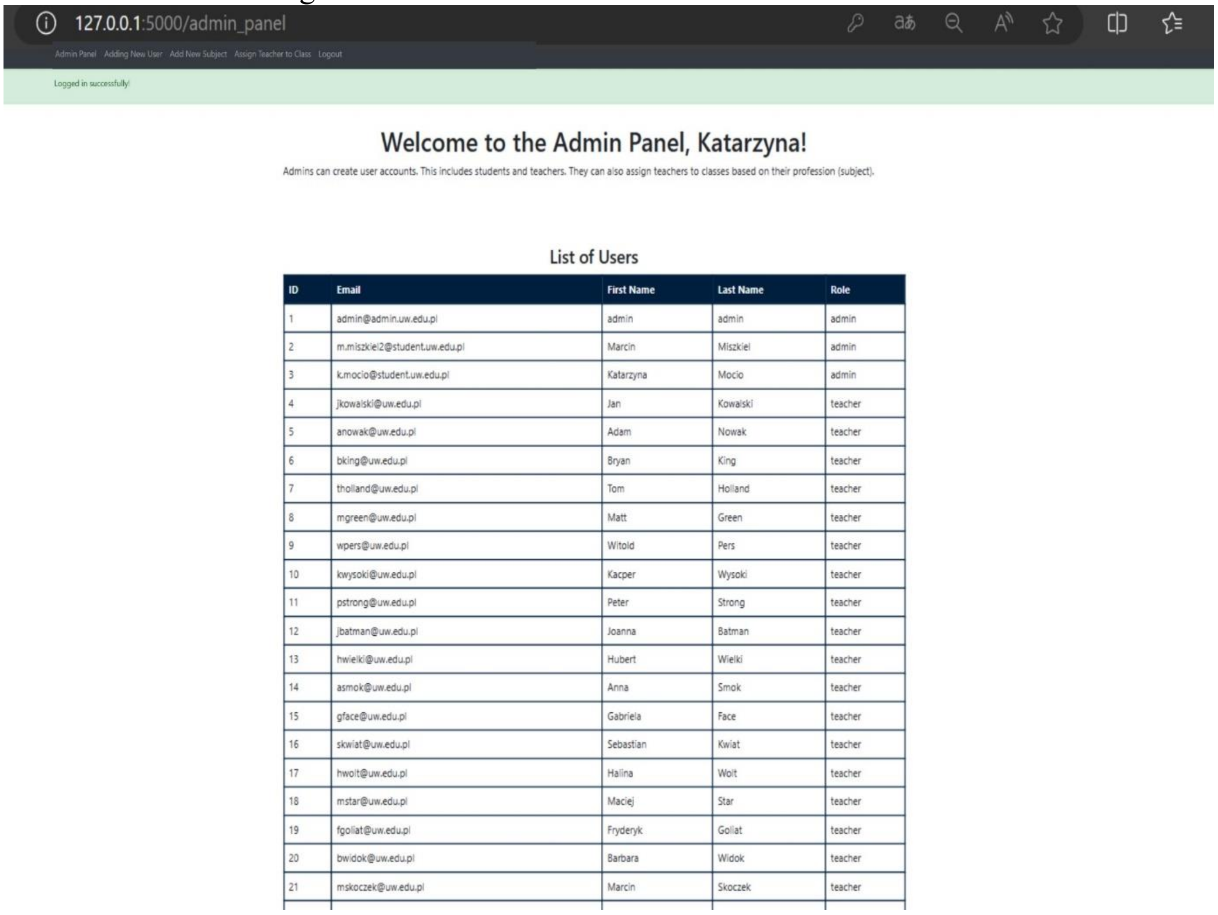
Source: Own work, based on Python, flask.

ADMINISTRATOR

ADMIN PANEL

The Admin Panel is the initial landing page upon logging in as an administrator. It encompasses tables presenting crucial information, including the list of application users, subjects, and classes. Furthermore, users benefit from a quick overview of teacher-class assignments and grades entered by educators.

Pic. 4. Admin Panel Page



Logged in successfully!

Welcome to the Admin Panel, Katarzyna!

Admins can create user accounts. This includes students and teachers. They can also assign teachers to classes based on their profession (subject).

List of Users

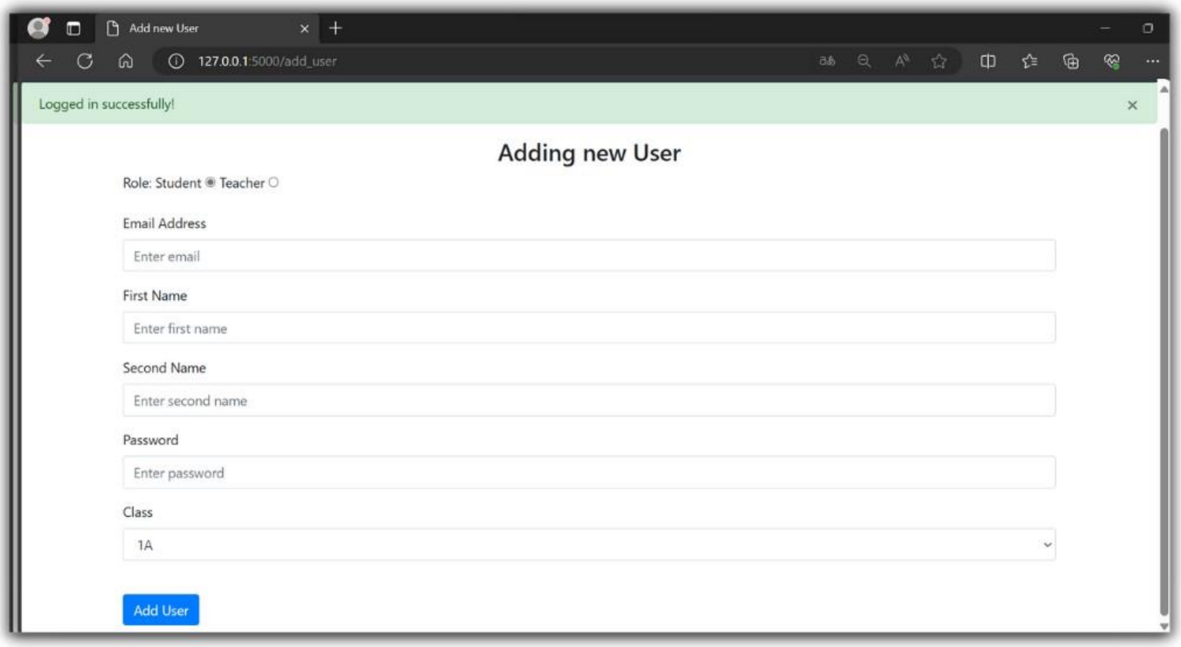
ID	Email	First Name	Last Name	Role
1	admin@admin.uw.edu.pl	admin	admin	admin
2	m.miszkiel2@student.uw.edu.pl	Marcin	Miszkiel	admin
3	k.mocio@student.uw.edu.pl	Katarzyna	Mocio	admin
4	jkowalski@uw.edu.pl	Jan	Kowalski	teacher
5	anowak@uw.edu.pl	Adam	Nowak	teacher
6	bking@uw.edu.pl	Bryan	King	teacher
7	tholland@uw.edu.pl	Tom	Holland	teacher
8	mgreen@uw.edu.pl	Matt	Green	teacher
9	wpers@uw.edu.pl	Witold	Pers	teacher
10	kwysoki@uw.edu.pl	Kacper	Wysoki	teacher
11	pstrong@uw.edu.pl	Peter	Strong	teacher
12	jbatman@uw.edu.pl	Joanna	Batman	teacher
13	hwielki@uw.edu.pl	Hubert	Wielki	teacher
14	asmok@uw.edu.pl	Anna	Smok	teacher
15	gface@uw.edu.pl	Gabriela	Face	teacher
16	skwiat@uw.edu.pl	Sebastian	Kwiat	teacher
17	hwolt@uw.edu.pl	Halina	Wolt	teacher
18	mstar@uw.edu.pl	Maciej	Star	teacher
19	fgoliat@uw.edu.pl	Fryderyk	Goliat	teacher
20	bwidok@uw.edu.pl	Barbara	Widok	teacher
21	mskoczek@uw.edu.pl	Marcin	Skoczek	teacher

Source: Own work, based on Python, flask, sqlite3.

ADD NEW USER

The administrator has the capability to add new users to the journal, both teachers and students. The selection of a user's role is made through an appropriate single-choice field. Subsequently, necessary information such as email, first name, last name, and password is provided. The entered data undergoes thorough verification to ensure compliance with established conventions. Additionally, when adding a teacher, it is mandatory to select the subject they will teach, and for a student, the choice of the class they will attend is required. Information about subjects and classes is current and dynamically retrieved from the database in real-time.

Pic. 5. Add New User Page



Logged in successfully!

Adding new User

Role: Student [✓ ●] Teacher ○

Email Address

Enter email

First Name

Enter first name

Second Name

Enter second name

Password

Enter password

Class

1A

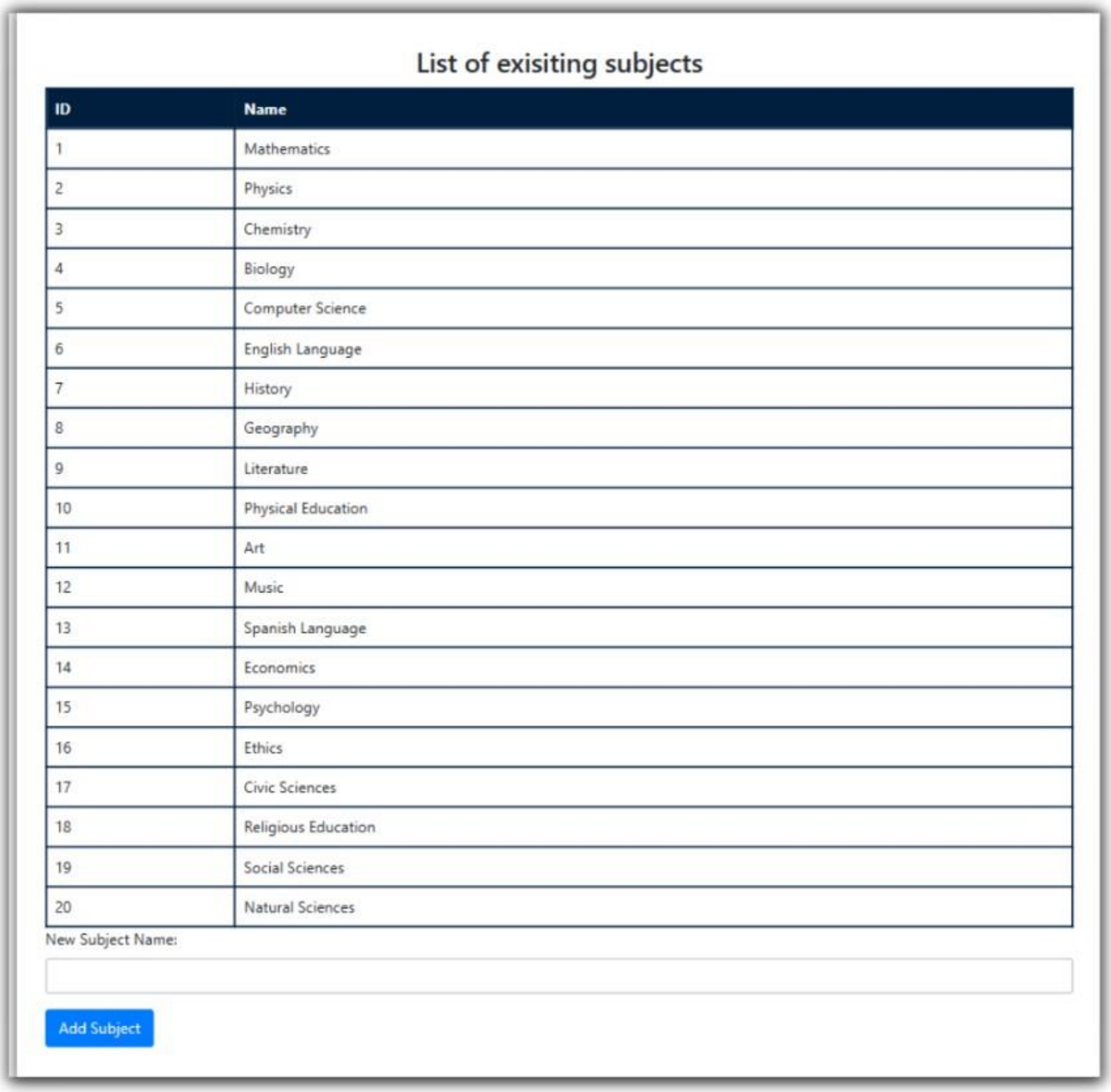
Add User

Source: Own work, based on Python, flask.

ADD NEW SUBJECT

There is also an option to add a new subject. The current list of subjects is displayed on the page. Below, it is possible to enter a new subject. During the verification of the new subject, both the correct convention and the absence of the subject in the database are checked. After entering the new subject, the list on the page refreshes, allowing the visibility of the newly added subject.

Pic. 6. Add New Subject Page



List of exisiting subjects

ID	Name
1	Mathematics
2	Physics
3	Chemistry
4	Biology
5	Computer Science
6	English Language
7	History
8	Geography
9	Literature
10	Physical Education
11	Art
12	Music
13	Spanish Language
14	Economics
15	Psychology
16	Ethics
17	Civic Sciences
18	Religious Education
19	Social Sciences
20	Natural Sciences

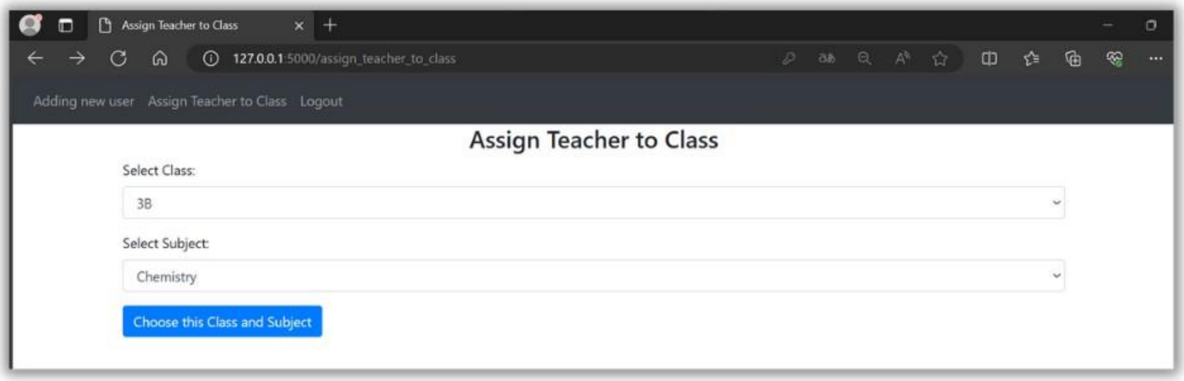
New Subject Name:

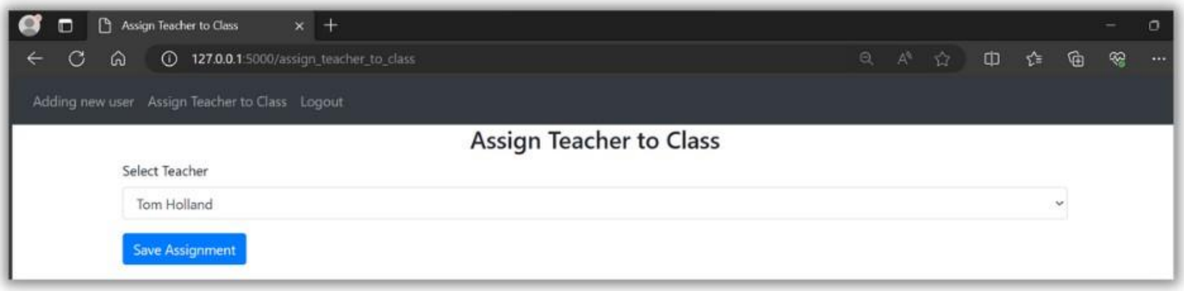
Source: Own work, based on Python, flask, sqlite3.

ASSIGN TEACHER TO CLASS

Assigning a teacher to a class is a two-step process. Initially, a combination of class and subject is selected. If no teacher exists for the specified combination, the assignment becomes feasible. Subsequently, another page is activated, allowing the selection of a teacher based on the list of educators instructing the previously chosen subject.

Pic. 7. Assign Teacher to Class Page





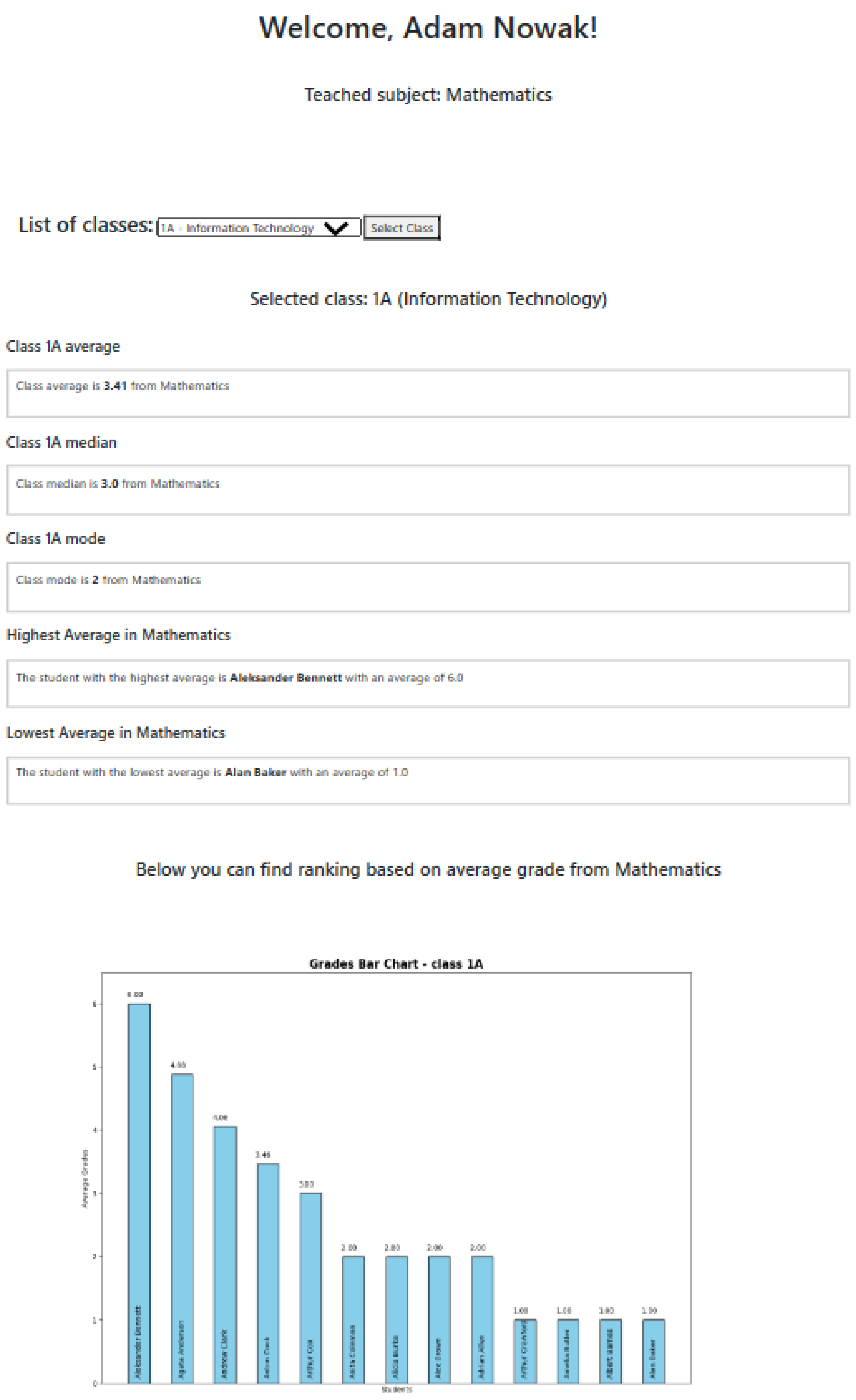
Source: Own work, based on Python, flask, sqlite3.

TEACHER

TEACHER DASHBOARD

In the teacher panel, user has access to the teacher dashboard, which contains basic statistics on grades in individual classes and a chart with the ranking of students based on the average grade in the subject taught by the logged-in teacher. At the top of the page there is a drop-down list from which the teacher selects one of the classes he teaches, and after pressing the "Select Class" button, the appropriate statistics and a chart are loaded below. The data is taken from the current database and therefore automatically takes into account also recently entered ratings or new students assigned to class.

Pic. 8. Teacher Dashboard Page



Welcome, Adam Nowak!

Teached subject: Mathematics

List of classes: 1A - Information Technology | Select Class

Selected class: 1A (Information Technology)

Class 1A average

Class average is **3.41** from Mathematics

Class 1A median

Class median is **3.0** from Mathematics

Class 1A mode

Class mode is **2** from Mathematics

Highest Average in Mathematics

The student with the highest average is **Aleksander Bennett** with an average of 6.0

Lowest Average in Mathematics

The student with the lowest average is **Alan Baker** with an average of 1.0

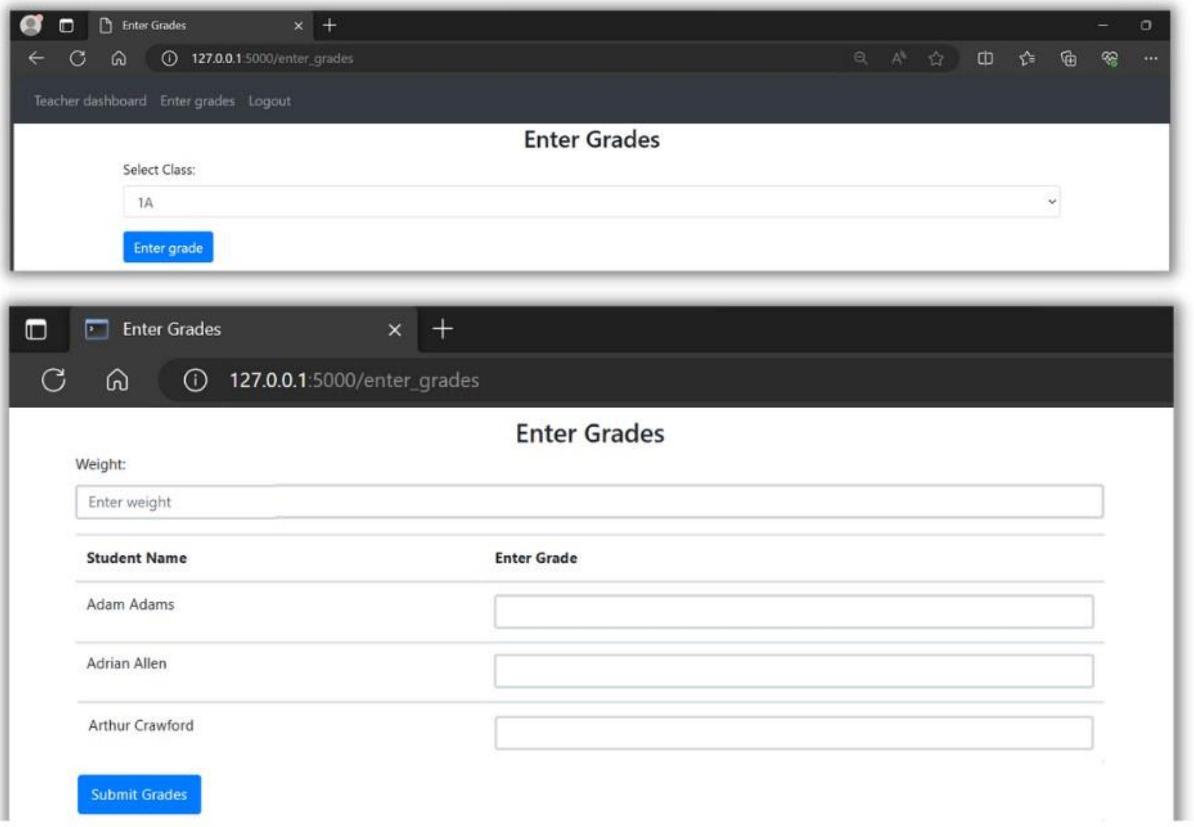
Below you can find ranking based on average grade from Mathematics

Source: Own work, based on Python, flask, sqlite3, matplotlib.

ENTER GRADES

The process of entering grades by teachers is a two-step procedure. Initially, the teacher selects the class for which they intend to input grades. On the subsequent page, the option to enter grades is displayed. The entire list of students in the selected class is visible. It is necessary to input the grade's weight within the range (0,1] and grades for the chosen students. It is not mandatory to enter grades for every student, and only the filled-in grades will be recorded in the database.

Pic. 9. Enter Grades Page



Enter Grades

Select Class: 1A

Enter Grades

Weight:

Student Name	Enter Grade
Adam Adams	
Adrian Allen	
Arthur Crawford	

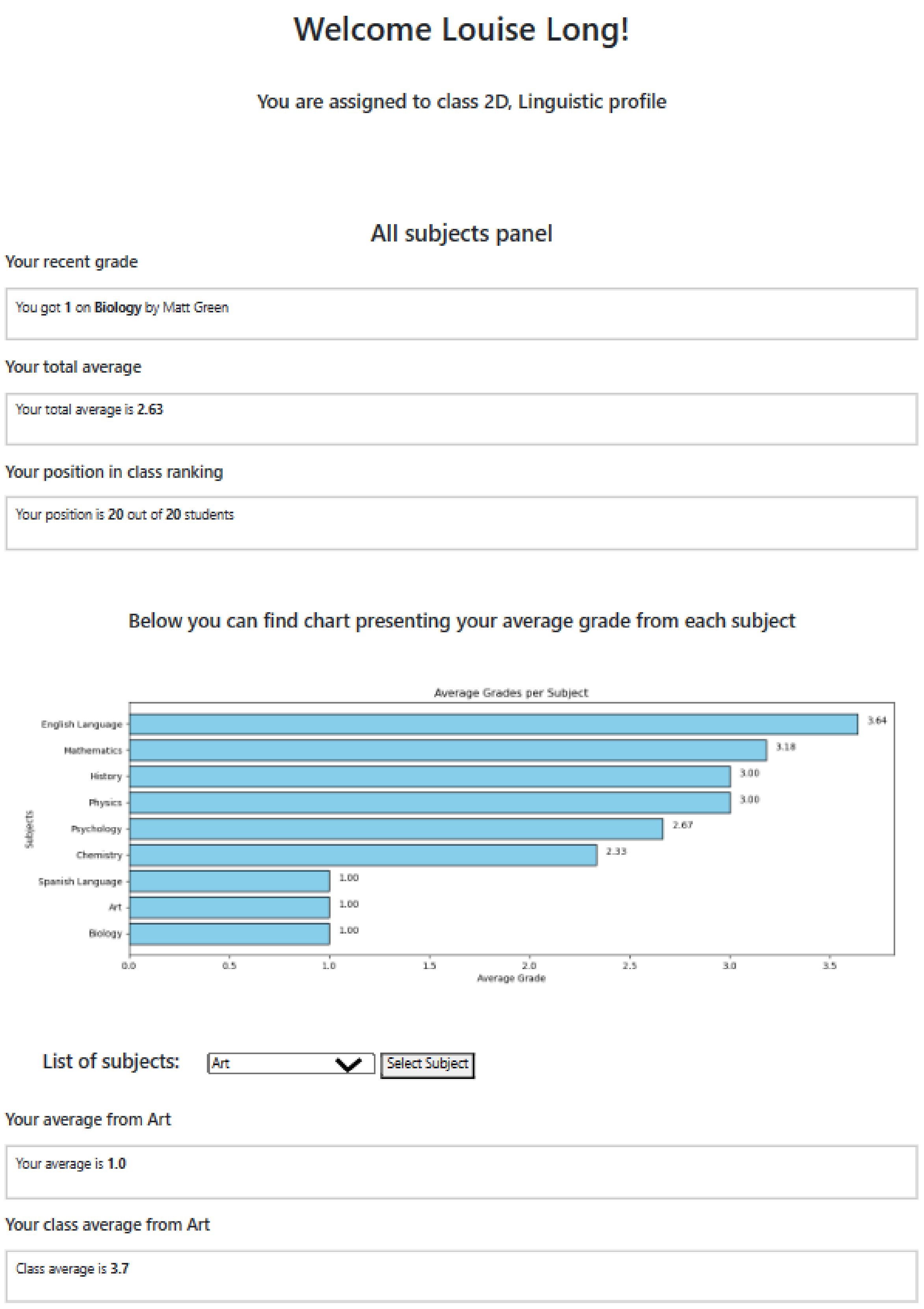
Source: Own work, based on Python, flask, sqlite3.

STUDENT

STUDENT DASHBOARD

In the student panel, user has access to the student dashboard, which contains basic statistics about his grades with chart presenting average grade from each subject he attends to. Below that, there is a drop-down list from which the student selects one of the subjects, and after pressing the "Select Subject" button, his and his class average grade from selected subject are displayed below.

Pic. 10. Student Dashboard Page



Welcome Louise Long!

You are assigned to class 2D, Linguistic profile

All subjects panel

Your recent grade

You got **1** on **Biology** by Matt Green

Your total average

Your total average is **2.63**

Your position in class ranking

Your position is **20** out of **20** students

Below you can find chart presenting your average grade from each subject

List of subjects: Art Select Subject

Your average from Art

Your average is **1.0**

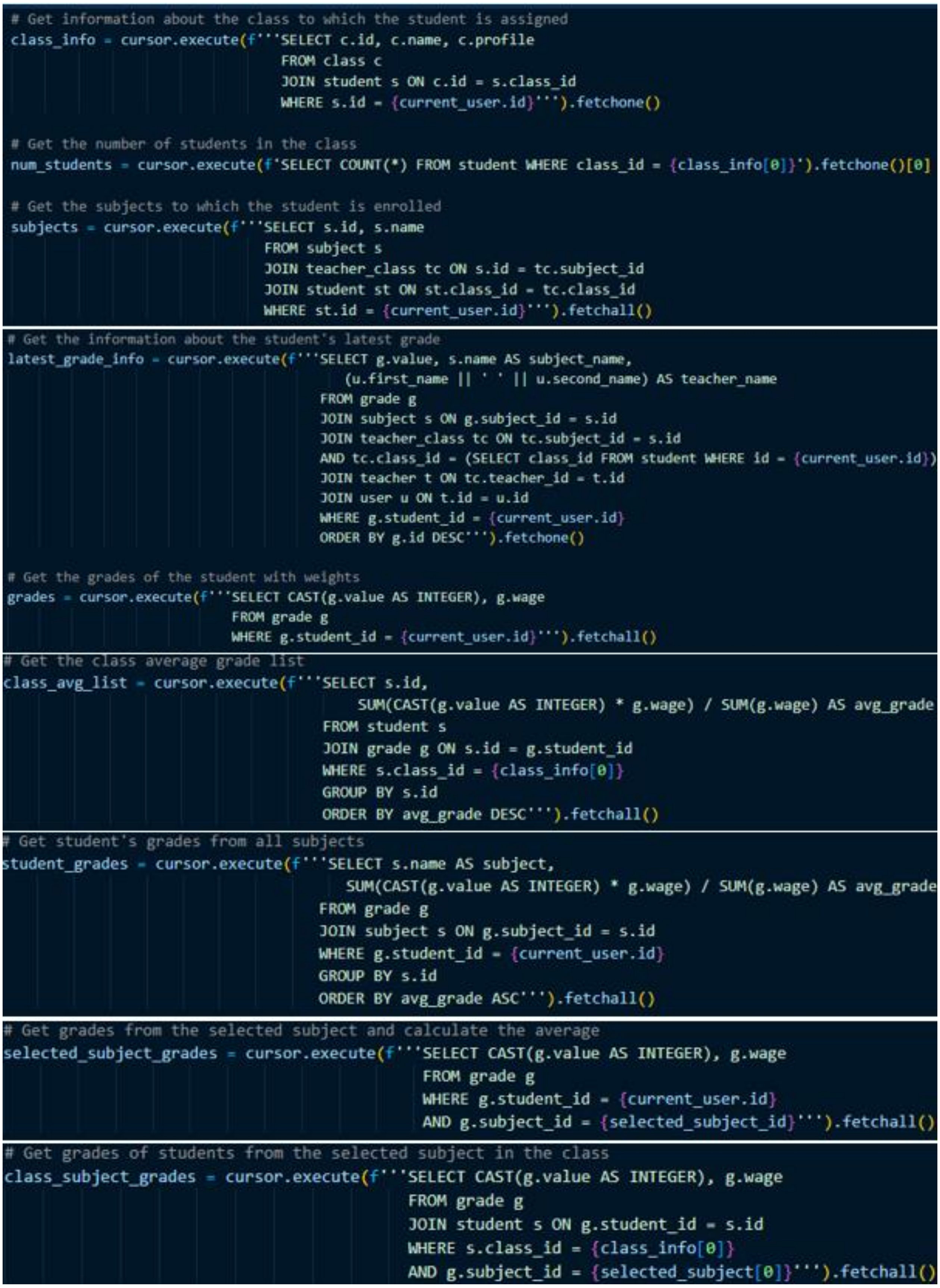
Your class average from Art

Class average is **3.7**

Source: Own work, based on Python, flask, sqlite3, matplotlib.

USED SQL QUERIES

Pic. 11. Queries part 1



```
# Get information about the class to which the student is assigned
class_info = cursor.execute(f'''SELECT c.id, c.name, c.profile
                                FROM class c
                                JOIN student s ON c.id = s.class_id
                                WHERE s.id = {current_user.id}''').fetchone()

# Get the number of students in the class
num_students = cursor.execute(f'SELECT COUNT(*) FROM student WHERE class_id = {class_info[0]}').fetchone()[0]

# Get the subjects to which the student is enrolled
subjects = cursor.execute(f'''SELECT s.id, s.name
                              FROM subject s
                              JOIN teacher_class tc ON s.id = tc.subject_id
                              JOIN student st ON st.class_id = tc.class_id
                              WHERE st.id = {current_user.id}''').fetchall()
```

```
# Get the information about the student's latest grade
latest_grade_info = cursor.execute(f'''SELECT g.value, s.name AS subject_name,
                                       (u.first_name || ' ' || u.second_name) AS teacher_name
                                       FROM grade g
                                       JOIN subject s ON g.subject_id = s.id
                                       JOIN teacher_class tc ON tc.subject_id = s.id
                                       AND tc.class_id = (SELECT class_id FROM student WHERE id = {current_user.id})
                                       JOIN teacher t ON tc.teacher_id = t.id
                                       JOIN user u ON t.id = u.id
                                       WHERE g.student_id = {current_user.id}
                                       ORDER BY g.id DESC''').fetchone()

# Get the grades of the student with weights
grades = cursor.execute(f'''SELECT CAST(g.value AS INTEGER), g.wage
                            FROM grade g
                            WHERE g.student_id = {current_user.id}''').fetchall()
```

```
# Get the class average grade list
class_avg_list = cursor.execute(f'''SELECT s.id,
                                    SUM(CAST(g.value AS INTEGER) * g.wage) / SUM(g.wage) AS avg_grade
                                    FROM student s
                                    JOIN grade g ON s.id = g.student_id
                                    WHERE s.class_id = {class_info[0]}
                                    GROUP BY s.id
                                    ORDER BY avg_grade DESC''').fetchall()
```

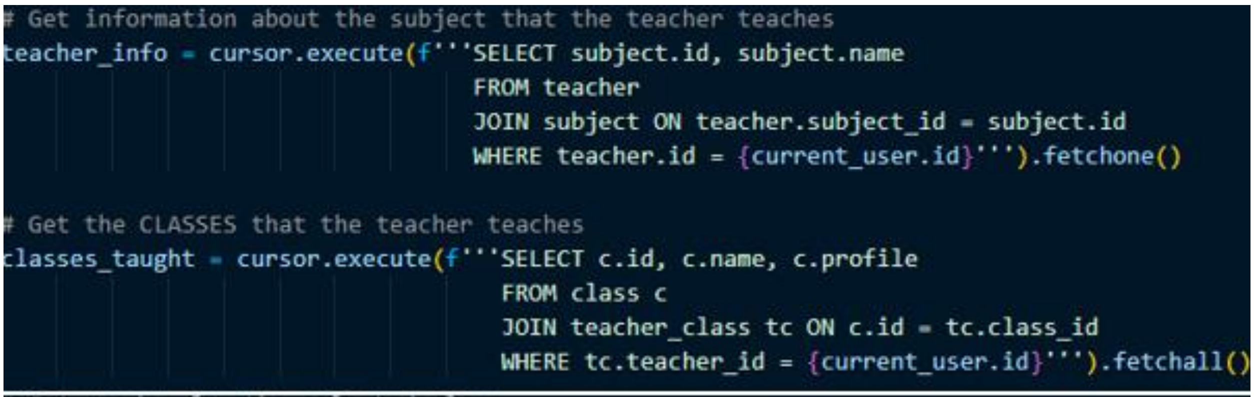
```
# Get student's grades from all subjects
student_grades = cursor.execute(f'''SELECT s.name AS subject,
                                    SUM(CAST(g.value AS INTEGER) * g.wage) / SUM(g.wage) AS avg_grade
                                    FROM grade g
                                    JOIN subject s ON g.subject_id = s.id
                                    WHERE g.student_id = {current_user.id}
                                    GROUP BY s.id
                                    ORDER BY avg_grade ASC''').fetchall()
```

```
# Get grades from the selected subject and calculate the average
selected_subject_grades = cursor.execute(f'''SELECT CAST(g.value AS INTEGER), g.wage
                                             FROM grade g
                                             WHERE g.student_id = {current_user.id}
                                             AND g.subject_id = {selected_subject_id}''').fetchall()
```

```
# Get grades of students from the selected subject in the class
class_subject_grades = cursor.execute(f'''SELECT CAST(g.value AS INTEGER), g.wage
                                          FROM grade g
                                          JOIN student s ON g.student_id = s.id
                                          WHERE s.class_id = {class_info[0]}
                                          AND g.subject_id = {selected_subject[0]}''').fetchall()
```

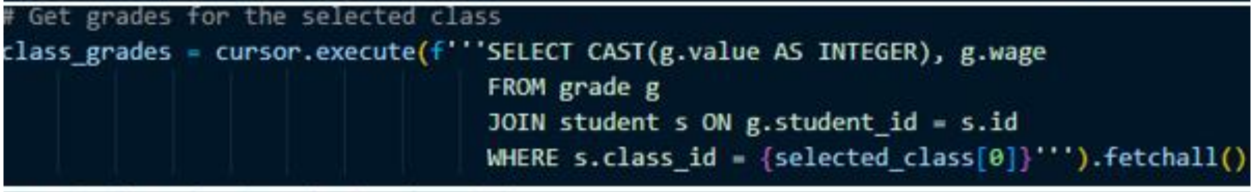
Source: Own work, based on Python, flask, sqlite3

Pic. 12. Queries part 2

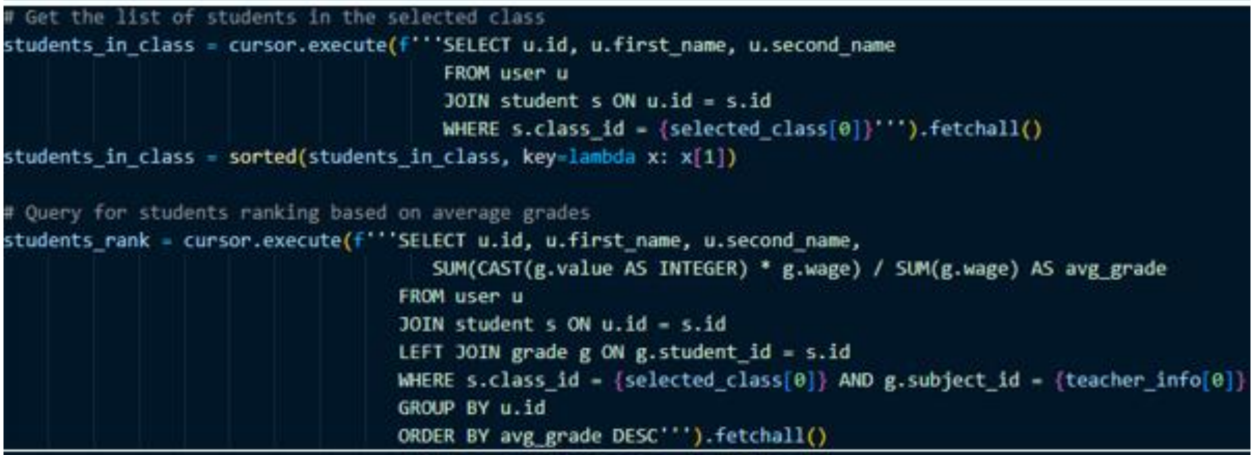


```
# Get information about the subject that the teacher teaches
teacher_info = cursor.execute(f'''SELECT subject.id, subject.name
                                  FROM teacher
                                  JOIN subject ON teacher.subject_id = subject.id
                                  WHERE teacher.id = {current_user.id}''').fetchone()

# Get the CLASSES that the teacher teaches
lasses_taught = cursor.execute(f'''SELECT c.id, c.name, c.profile
                                    FROM class c
                                    JOIN teacher_class tc ON c.id = tc.class_id
                                    WHERE tc.teacher_id = {current_user.id}''').fetchall()
```

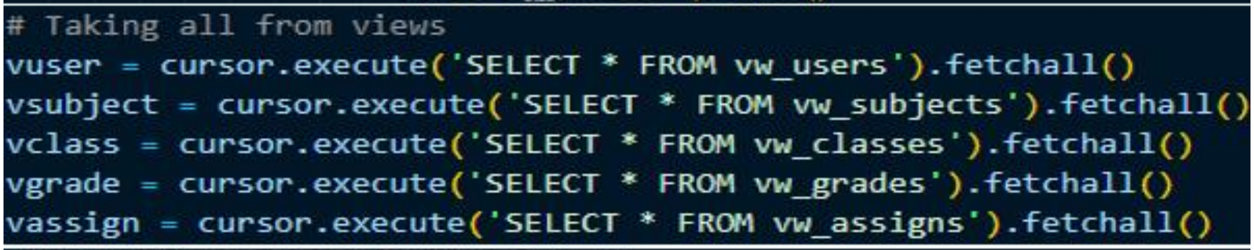


```
# Get grades for the selected class
lass_grades = cursor.execute(f'''SELECT CAST(g.value AS INTEGER), g.wage
                                  FROM grade g
                                  JOIN student s ON g.student_id = s.id
                                  WHERE s.class_id = {selected_class[0]}''').fetchall()
```

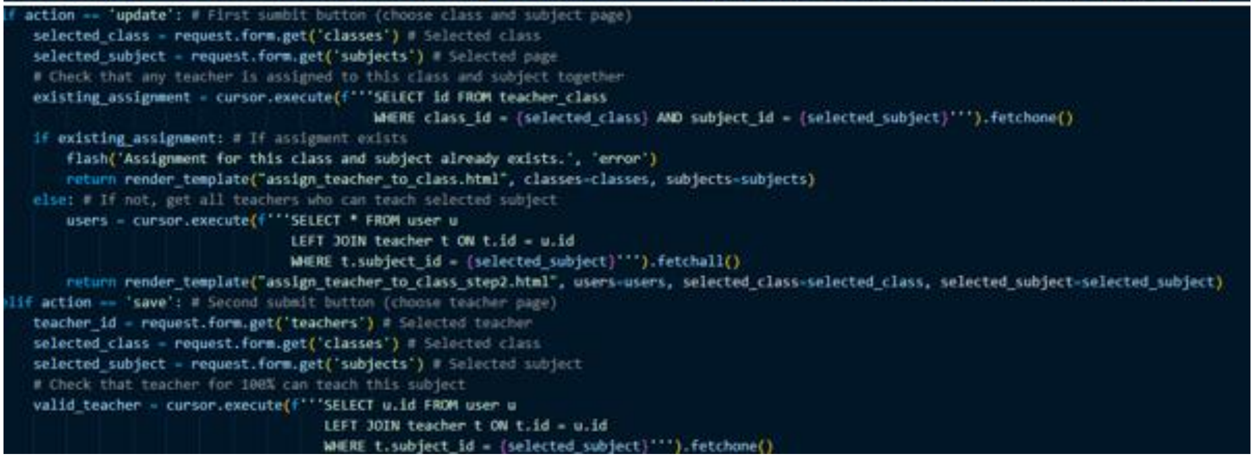


```
# Get the list of students in the selected class
students_in_class = cursor.execute(f'''SELECT u.id, u.first_name, u.second_name
                                       FROM user u
                                       JOIN student s ON u.id = s.id
                                       WHERE s.class_id = {selected_class[0]}''').fetchall()
students_in_class = sorted(students_in_class, key=lambda x: x[1])

# Query for students ranking based on average grades
students_rank = cursor.execute(f'''SELECT u.id, u.first_name, u.second_name,
                                     SUM(CAST(g.value AS INTEGER) * g.wage) / SUM(g.wage) AS avg_grade
                                   FROM user u
                                   JOIN student s ON u.id = s.id
                                   LEFT JOIN grade g ON g.student_id = s.id
                                   WHERE s.class_id = {selected_class[0]} AND g.subject_id = {teacher_info[0]}
                                   GROUP BY u.id
                                   ORDER BY avg_grade DESC''').fetchall()
```



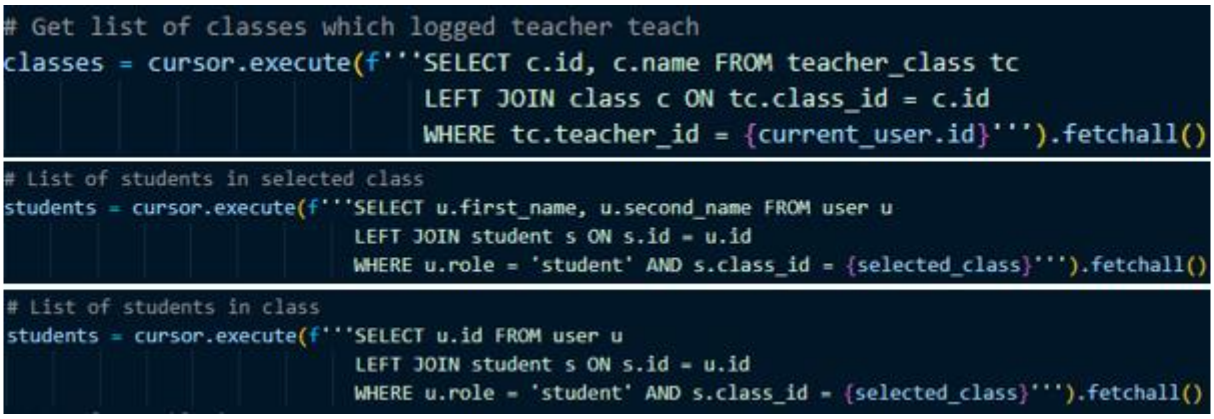
```
# Taking all from views
vuser = cursor.execute('SELECT * FROM vw_users').fetchall()
vsubject = cursor.execute('SELECT * FROM vw_subjects').fetchall()
vclass = cursor.execute('SELECT * FROM vw_classes').fetchall()
vgrade = cursor.execute('SELECT * FROM vw_grades').fetchall()
vassign = cursor.execute('SELECT * FROM vw_assigns').fetchall()
```



```
f action == 'update': # First sumbit button (choose class and subject page)
   selected_class = request.form.get('classes') # Selected class
   selected_subject = request.form.get('subjects') # Selected page
   # Check that any teacher is assigned to this class and subject together
   existing_assignment = cursor.execute(f'''SELECT id FROM teacher_class
                                            WHERE class_id = {selected_class} AND subject_id = {selected_subject}''').fetchone()
   if existing_assignment: # If assigment exists
       flash('Assignment for this class and subject already exists.', 'error')
       return render_template("assign_teacher_to_class.html", classes=classes, subjects=subjects)
   else: # If not, get all teachers who can teach selected subject
       users = cursor.execute(f'''SELECT * FROM user u
                                  LEFT JOIN teacher t ON t.id = u.id
                                  WHERE t.subject_id = {selected_subject}''').fetchall()
       return render_template("assign_teacher_to_class_step2.html", users=users, selected_class=selected_class, selected_subject=selected_subject)
lif action == 'save': # Second submit button (choose teacher page)
   teacher_id = request.form.get('teachers') # Selected teacher
   selected_class = request.form.get('classes') # Selected class
   selected_subject = request.form.get('subjects') # Selected subject
   # Check that teacher for 100% can teach this subject
   valid_teacher = cursor.execute(f'''SELECT u.id FROM user u
                                      LEFT JOIN teacher t ON t.id = u.id
                                      WHERE t.subject_id = {selected_subject}''').fetchone()
```

Source: Own work, based on Python, sqlite3.

Pic. 13. Queries part 3



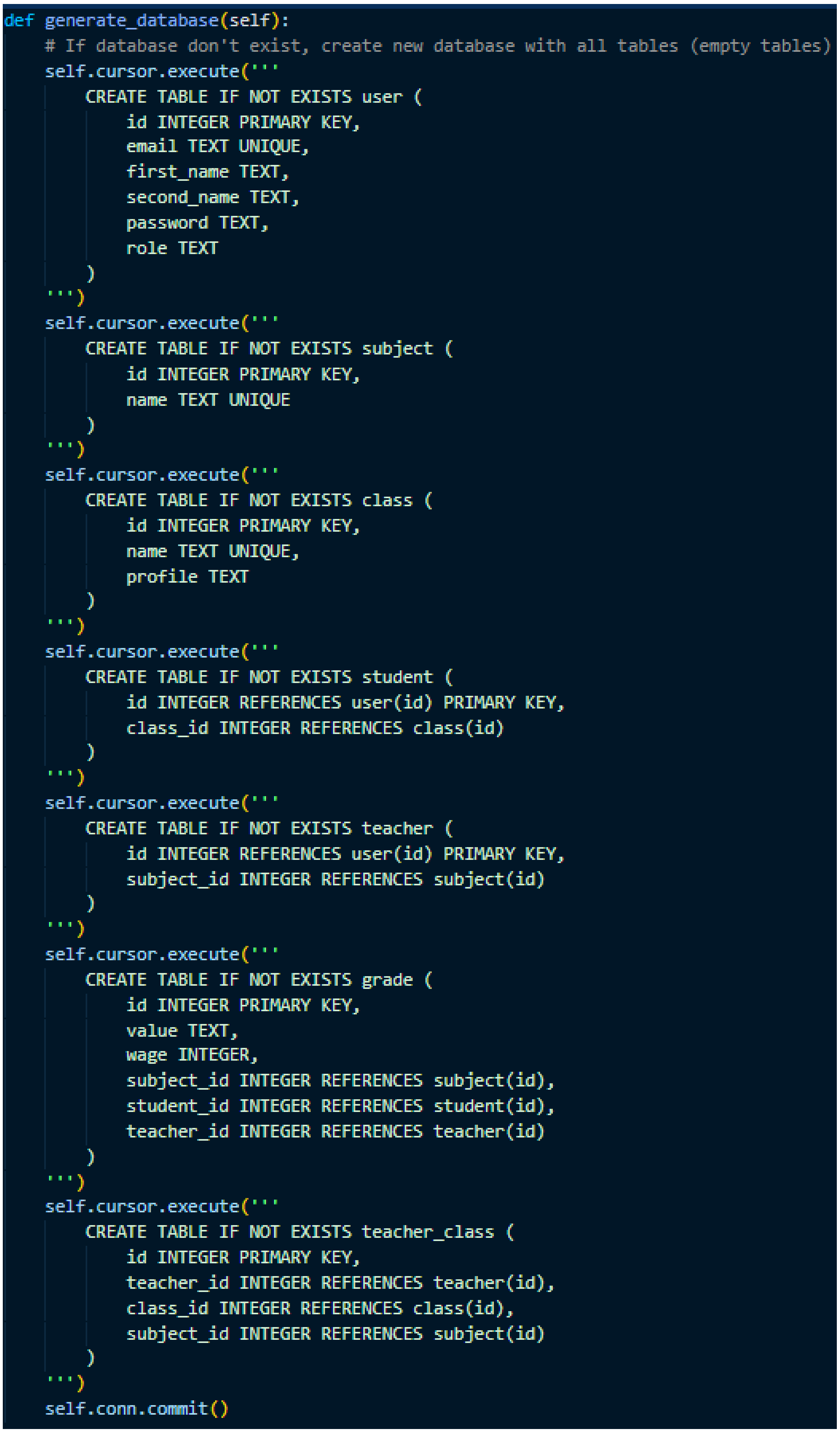
```
# Get list of classes which logged teacher teach
classes = cursor.execute(f'''SELECT c.id, c.name FROM teacher_class tc
                             LEFT JOIN class c ON tc.class_id = c.id
                             WHERE tc.teacher_id = {current_user.id}''').fetchall()
```

```
# List of students in selected class
students = cursor.execute(f'''SELECT u.first_name, u.second_name FROM user u
                              LEFT JOIN student s ON s.id = u.id
                              WHERE u.role = 'student' AND s.class_id = {selected_class}''').fetchall()
```

```
# List of students in class
students = cursor.execute(f'''SELECT u.id FROM user u
                              LEFT JOIN student s ON s.id = u.id
                              WHERE u.role = 'student' AND s.class_id = {selected_class}''').fetchall()
```

Source: Own work, based on Python, sqlite3.

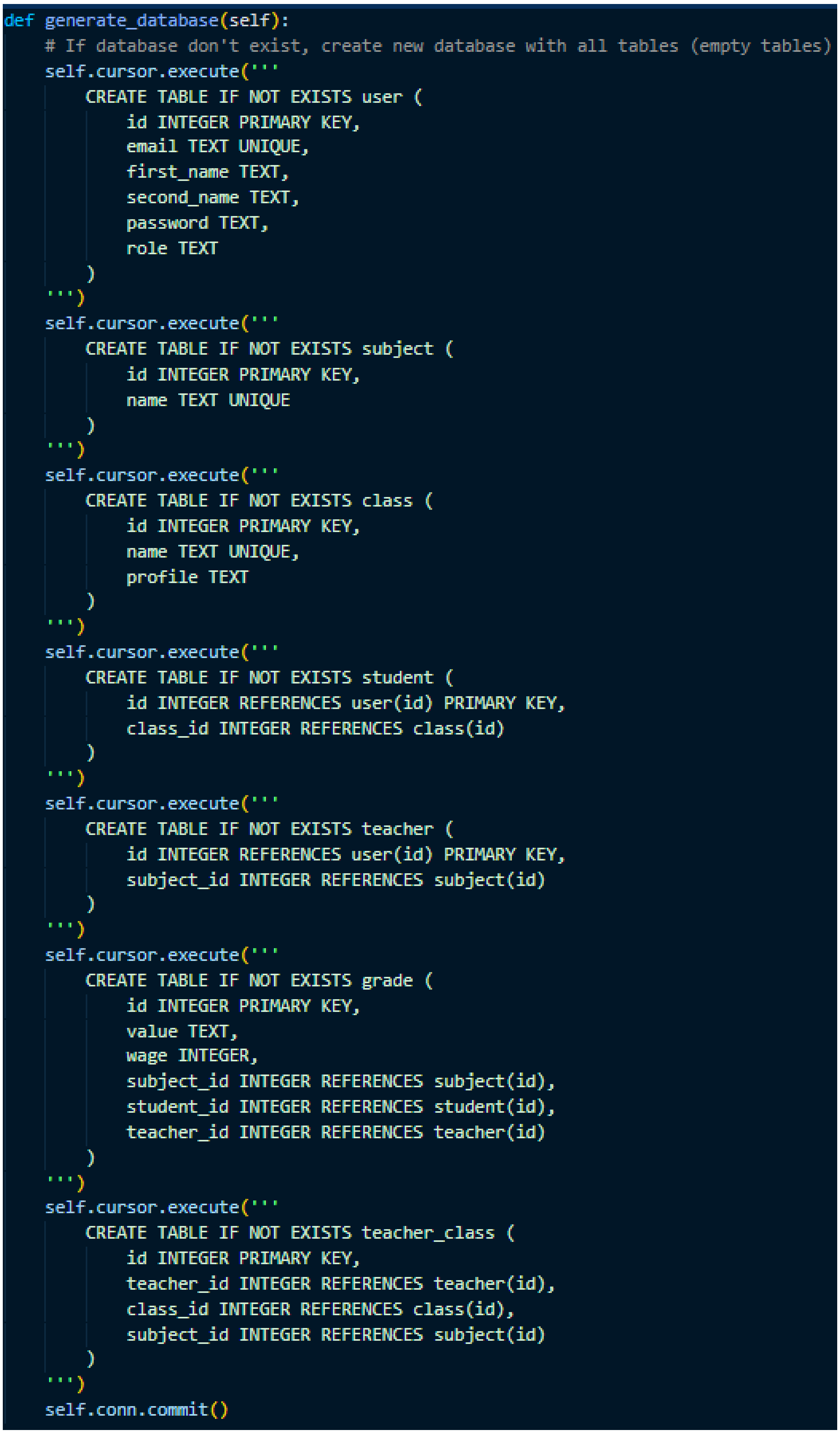
Pic. 14. Create Database class method



```
def generate_database(self):
    # If database don't exist, create new database with all tables (empty tables)
    self.cursor.execute('''
        CREATE TABLE IF NOT EXISTS user (
            id INTEGER PRIMARY KEY,
            email TEXT UNIQUE,
            first_name TEXT,
            second_name TEXT,
            password TEXT,
            role TEXT
        )
    ''')
    self.cursor.execute('''
        CREATE TABLE IF NOT EXISTS subject (
            id INTEGER PRIMARY KEY,
            name TEXT UNIQUE
        )
    ''')
    self.cursor.execute('''
        CREATE TABLE IF NOT EXISTS class (
            id INTEGER PRIMARY KEY,
            name TEXT UNIQUE,
            profile TEXT
        )
    ''')
    self.cursor.execute('''
        CREATE TABLE IF NOT EXISTS student (
            id INTEGER REFERENCES user(id) PRIMARY KEY,
            class_id INTEGER REFERENCES class(id)
        )
    ''')
    self.cursor.execute('''
        CREATE TABLE IF NOT EXISTS teacher (
            id INTEGER REFERENCES user(id) PRIMARY KEY,
            subject_id INTEGER REFERENCES subject(id)
        )
    ''')
    self.cursor.execute('''
        CREATE TABLE IF NOT EXISTS grade (
            id INTEGER PRIMARY KEY,
            value TEXT,
            wage INTEGER,
            subject_id INTEGER REFERENCES subject(id),
            student_id INTEGER REFERENCES student(id),
            teacher_id INTEGER REFERENCES teacher(id)
        )
    ''')
    self.cursor.execute('''
        CREATE TABLE IF NOT EXISTS teacher_class (
            id INTEGER PRIMARY KEY,
            teacher_id INTEGER REFERENCES teacher(id),
            class_id INTEGER REFERENCES class(id),
            subject_id INTEGER REFERENCES subject(id)
        )
    ''')
    self.conn.commit()
```

Source: Own work, based on Python, sqlite3.

Pic. 15. Create views class method

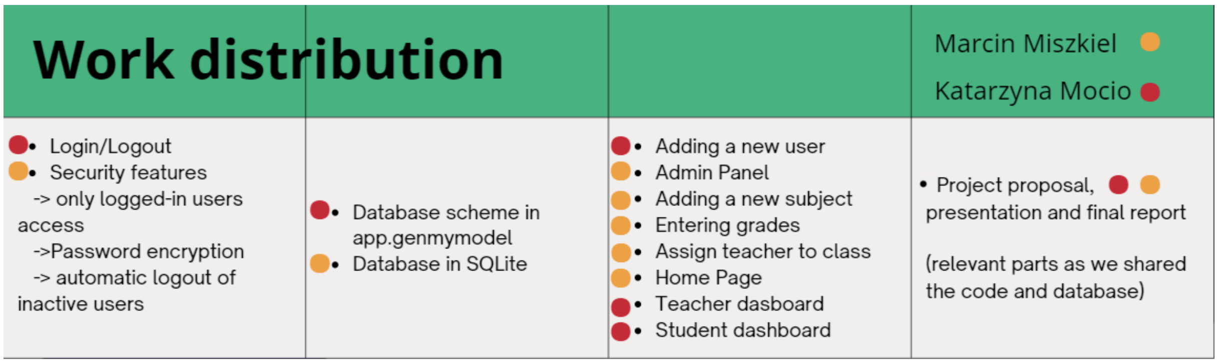


```
def generate_database(self):
    # If database don't exist, create new database with all tables (empty tables)
    self.cursor.execute('''
        CREATE TABLE IF NOT EXISTS user (
            id INTEGER PRIMARY KEY,
            email TEXT UNIQUE,
            first_name TEXT,
            second_name TEXT,
            password TEXT,
            role TEXT
        )
    ''')
    self.cursor.execute('''
        CREATE TABLE IF NOT EXISTS subject (
            id INTEGER PRIMARY KEY,
            name TEXT UNIQUE
        )
    ''')
    self.cursor.execute('''
        CREATE TABLE IF NOT EXISTS class (
            id INTEGER PRIMARY KEY,
            name TEXT UNIQUE,
            profile TEXT
        )
    ''')
    self.cursor.execute('''
        CREATE TABLE IF NOT EXISTS student (
            id INTEGER REFERENCES user(id) PRIMARY KEY,
            class_id INTEGER REFERENCES class(id)
        )
    ''')
    self.cursor.execute('''
        CREATE TABLE IF NOT EXISTS teacher (
            id INTEGER REFERENCES user(id) PRIMARY KEY,
            subject_id INTEGER REFERENCES subject(id)
        )
    ''')
    self.cursor.execute('''
        CREATE TABLE IF NOT EXISTS grade (
            id INTEGER PRIMARY KEY,
            value TEXT,
            wage INTEGER,
            subject_id INTEGER REFERENCES subject(id),
            student_id INTEGER REFERENCES student(id),
            teacher_id INTEGER REFERENCES teacher(id)
        )
    ''')
    self.cursor.execute('''
        CREATE TABLE IF NOT EXISTS teacher_class (
            id INTEGER PRIMARY KEY,
            teacher_id INTEGER REFERENCES teacher(id),
            class_id INTEGER REFERENCES class(id),
            subject_id INTEGER REFERENCES subject(id)
        )
    ''')
    self.conn.commit()
```

Source: Own work, based on Python, sqlite3.

WORK DISTRIBUTION

Pic. 15. Work distribution



Work distribution			Marcin Miszkiel 🟠 Katarzyna Mocio 🔴
🔴• Login/Logout 🟠• Security features -> only logged-in users access ->Password encryption -> automatic logout of inactive users	🔴• Database scheme in app.genmymodel 🟠• Database in SQLite	🔴• Adding a new user 🟠• Admin Panel 🟠• Adding a new subject 🟠• Entering grades 🟠• Assign teacher to class 🟠• Home Page 🔴• Teacher dasboard 🔴• Student dashboard	• Project proposal, 🔴 🟠 presentation and final report (relevant parts as we shared the code and database)

Source: Own work.