



ORACLE PROJECT

Project report

Supervised by :
Prof.mohammed Airaj

Prepared by :
zakariae Zaoui

Projet_Oracle

Concept

In summary, the goal of this project is to provide a secure web platform with a variety of user profiles. The platform should be created using the programming language of your choice and connected to an Oracle data base. There will be two users in the database: an administrator with full access rights and a regular user with access limits. To ensure that the access limits are followed, the users' connections will be tested. To record the acts taken by each user, journalization files will finally be generated.

Introduction

This platform is an educational management system that displays the notes of students. This system provides administrators with a friendly interface for managing student information such as name, notes, and subjects. The use of multiple user profiles ensures secure access. A data administrator (prof) has access to all of the data and has the ability to modify it. The students themselves can only view their personal information without having the option to change it.

Our goal is to provide a secure web platform with a straightforward connection that includes a connection button and two text fields for the user's name and password. The platform will be linked to an Oracle data base.

After that we are going to get a web page that contains the list of marks and subjects of the user and if its the admin it gives him the right to modify the marks

We'll test the access to our platform by concurrently connecting the two users using Node.js to make sure that the connection adheres to the proper access rights for the type of user. we are going to use thunder client to test the requests before testing our web platform.

Architecture :



Here are the general steps for connecting from the platform to the data base:

- Configuring the connection parameters: This includes specifying the connection details, which includes the user name and password. Charge the Oracle driver for the client platform: The Oracle Connection Pilote is a library that enables one to connect to the Oracle Database, often known as Oracledb for Node.js
- Establishing a connection The client platform can establish a secure connection with the Oracle data base by using the connection information and the Oracle pilot, according on the entered profile.
- Execute queries: After establishing a connection, the client platform can run SQL queries to interact with the data stored in the Oracle database.
- Terminate the connection: Terminating the connection will free up the platform's used resources when it is no longer necessary for it to connect to the database.

Oracle :

Oracle is a popular relational database management system (RDBMS) used for managing and storing large amounts of structured data. It offers advanced features such as security, reliability, scalability, and data recovery, making it a popular choice for businesses and organizations. Oracle is known for its flexible architecture, allowing it to run on multiple operating systems and providing compatibility with different programming languages.

after setting up oracle sata base management in our local machine we move to create the tables and the users

Tables creation :

users Table

In this project, a data base that was created with system users will be used as the basis for work. creation of the "student" table with the three columns "student id," "name," and "email." The primary key column "student id" is defined as primary key it can not be null

```
CREATE TABLE student (  
  student_id PRIMARY KEY,  
  name varchar(30) NOT NULL,  
  email varchar(30) NOT NULL  
);
```

users Table

the creation of the "notes" table with the four columns "note id," "student id," "subject," and "grade." In reference to the column "student id" of table "student," column "note id" is defined as the primary key, and column "student id" is defined as the foreign key.

```
CREATE TABLE notes (  
  note_id number(10) PRIMARY KEY,  
  student_id number(10) NOT NULL,  
  FOREIGN KEY (student_id) REFERENCES student(student_id)  
  
  subject varchar(30) NOT NULL,  
  grade number(10) NOT NULL  
);
```

Users creation :

Here we have to create two users a simple user and an administrator ,to create them we are going to follow these steps :

- we have to connect to the Oracle database as a user with sufficient privileges to create new users, such as the SYSTEM user. wich is already done.
- we run the following SQL command to create a new user:

```
CREATE USER user1 IDENTIFIED BY user1 default tablespace users;  
CREATE USER admin IDENTIFIED BY admin default tablespace users;
```

- now we need to grant the necessary privileges to these users, For example, to grant the CREATE SESSION privilege

Grant privileges :

we need to allow "admin" user access to all privileges.

```
grant all privileges to admin;
```

Granting the user "user1" access to the "new session" and "select" functions on the "student" and "notes" database.

```
GRANT CREATE SESSION TO user1;  
GRANT SELECT ON student TO user1;  
GRANT SELECT ON notes TO user1;
```

Testing :

for the first test we are going to use our simple user "user1" to do that we are going to run a simple SQL query such as :

```
INSERT INTO notes (note_id,student_id,subject,grade)  
VALUES (1, 1, "oracle",19);
```

we obtain an error because we didn't give the user "user1" the privilege to insert any value in notes table

Erreur SQL : ORA-01031: privilèges insuffisants

ORA-01031. 00000 - "insufficient privileges"

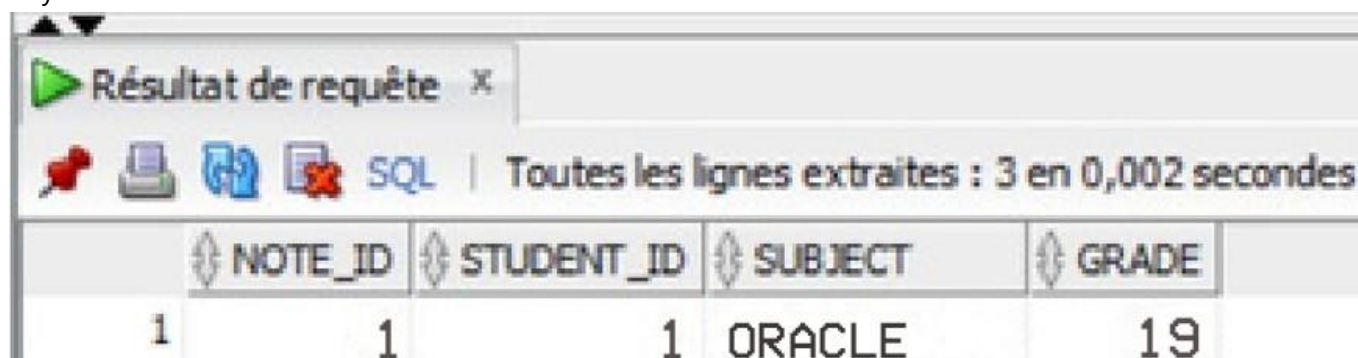
Cause: An attempt was made to perform a database operation without the necessary privileges.

Action: Ask your database administrator or designated security administrator to grant you the necessary privileges

now we are going to do the same process but with admin privileges

```
INSERT INTO notes (note_id,student_id,subject,grade)  
VALUES (1, 1, "oracle",19);
```

as you can see we were able to insert a value in the notes table



The screenshot shows a window titled "Résultat de requête" with a toolbar containing icons for a pin, printer, copy, and delete, along with an "SQL" label. Below the toolbar, it states "Toutes les lignes extraites : 3 en 0,002 secondes". The main area displays a table with the following data:

	NOTE_ID	STUDENT_ID	SUBJECT	GRADE
1	1	1	ORACLE	19

This is a simple example, using "admin" as an admin and "user1" as a normal user, you can perform various privileges on other types of users.

WEB Platform test :

Overview

Our code is an example of a server-side Web application that implements login and note-display functionality.

- An authentication form where the user can enter their user id and password is displayed when they access the home page using the HTTP GET method. User submission of the form triggers the HTTP POST / login method to be used. This method compares the user's identity information with the list of authorized users by extracting it from the body of the request. A notice of login failure is sent if the user cannot be found in the list. Otherwise, the connection is recorded in a journal file, and the user is directed to the appropriate page according to his or her user name.
- A group of SQL queries are executed on an Oracle database when a user accesses a web page using the HTTP GET method. Each query's result is converted to HTML for a table display on the page.

Express Server

This code sets up an Express.js web application which provides a simple login form for users. The express and oracledb npm modules are imported for use in the application. The fs module is imported for writing logs to a file.

The middleware functions `express.json()` and `express.urlencoded({ extended: true })` are applied to parse incoming JSON and URL-encoded data. The `allowedUsers` array contains a list of users who are allowed to log in.

The login form is created using HTML and is served as the response to a GET request to the root route `/`. On form submit, a POST request to `/login` is made. The username and password values are extracted from the request body. The code then tries to find a user in the `allowedUsers` array with the same username and password values. If the user is not found, a "Login Failed" message is sent as the response.

If a user is found, the code then checks if the user's username is "admin". If so, the response will redirect to `/admin`, otherwise it will redirect to `/normal`. A log message is written to a file `logs.txt` containing the date, time, and the username of the user who logged in.

Finally, the `/admin` route is defined to serve a table containing data from two tables in an Oracle database. The `oracledb.getConnection` function is used to connect to the Oracle database using the specified credentials. Two SQL queries are executed using the `connection.execute` method and the results are transformed into HTML tables which are sent as the response to the client. The HTML tables are styled using CSS.

```
const express = require('express');
const oracledb = require('oracledb');
const fs = require('fs');

const app = express();

app.use(express.json());
app.use(express.urlencoded({ extended: true }));

const allowedUsers = [
  {
    username: 'admin',
    password: 'admin',
    schema: 'schema1'
  },
  {
    username: 'user1',
    password: 'user1',
    schema: 'schema2'
  }
];

app.get('/', (req, res) => {
  res.send(`
    <head><style>
    form {
      display: flex;
      flex-direction: column;
      align-items: left;
      padding: 20px;
```

```

    }
    input[type="text"],
input[type="password"] {
width: 200px;    height:
40px;    margin-bottom:
20px;    padding-left:
10px;    font-size: 16px;
border: 1px solid pink;
border-radius: 5px;
    }
button[type="submit"] {
width: 200px;    height:
40px;    background-color:
pink;    color: white;
font-size: 16px;
border: none;    border-
radius: 5px;    cursor:
pointer;
    }
</style>
</head>
    <form action="/login" method="post">
        <input type="text" name="username" placeholder="Username">
        <input type="password" name="password" placeholder="Password">
        <button type="submit">Submit</button>
    </form>
`);
});

app.post('/login', async (req, res) => {
let connection;    const { username,
password } = req.body;
    const user = allowedUsers.find(u => u.username === username && u.password
=== password);
    if (!user) {    return
res.send('Login Failed');
    }    if (username ===
'admin') {
res.redirect('/admin');
    } else {
res.redirect('/normal');
    }    const log = `[${new Date().toString()}] User ${username} logged
in\n`;    fs.appendFile('logs.txt', log, (err) => {    if (err) throw
err;
    console.log('The log was saved!');
});
});

```



```

});    app.get('/admin', async(req, res) => {
let connection;    try {        connection =
await oracledb.getConnection({        user:
"admin",        password: "admin",
connectString: "localhost/orcl"
});    }    const sqlQueries = [ "select * from projet.student",
"select * from projet.notes"];
// const sqlQueries = [ "select * from projet.car"];
const tables = [];    for (const sql of sqlQueries) {        const result =
await connection.execute(sql);        const tableHTML = result.rows.map(row
=> {            let rowHTML = '';            for (let i = 0; i <
result.metaData.length; i++) {                rowHTML += `<td>${row[i]}</td>`;
            }            return
`<tr>${rowHTML}</tr>`;
        }).join('');        tables.push({ metaData:
result.metaData, tableHTML });
    }
    res.send(`

    </form>
    <head><style>
button[type="submit"] {
align-items: center;
width: 200px;        height:
40px;        background-color:
pink;        color: white;
font-size: 16px;
border: none;        border-
radius: 5px;        cursor:
pointer;
    }    table {
width: 60%;        margin: 20px ;
border-collapse: collapse;
    }    th, td {
border: 2px solid pink;
padding: 15px;        text-
align: left;
    }    th {
background-color: lightpink;

```



```

        font-weight: bold;
    }
    tr:nth-
child(even) {
        background-color: lightpink;
    }
</style>
</head>

<h1>Tables</h1>
${tables.map(table => `
    <table>
        <thead>
            <tr>
                ${table.metaData.map(col => `<th>${col.name}</th>`).join('')}
            </tr>
        </thead>
        <tbody>
            ${table.tableHTML}
        </tbody>
    </table>

`).join('')}
<form action="/logout" method="post">
<button type="submit">logout</button>`);
    } catch (error) {
console.error(error);                res.send('Error
Occurred');
    } finally {
    }
});

app.get('/user', async(req, res) => {
let connection;    try {                connection =
await oracledb.getConnection({                user:
'user1',                password: 'user1',
connectString: "localhost/orcl"
    });                const sqlQueries = [ "select *
from projet.notes" ];
    const tables = [];    for (const sql of
sqlQueries) {        const result = await
connection.execute(sql);        const tableHTML =
result.rows.map(row => {            let rowHTML = '';
for (let i = 0; i < result.metaData.length; i++) {
rowHTML += `<td>${row[i]}</td>`;
            }            return
`<tr>${rowHTML}</tr>`;

```



```

    }).join('');
    tables.push({ metaData: result.metaData, tableHTML });
  }
  res.send(`    </form>
<head><style>
button[type="submit"] {
align-items: center;
width: 200px;    height:
40px;    background-
color: red;    color:
white;    font-size:
16px;    border: none;
border-radius: 5px;
cursor: pointer;
}    table {        width:
60%;        margin: 20px ;
border-collapse: collapse;
        }        th, td {
border: 1px solid red;
padding: 10px;        text-
align: left;
        }        th {
background-color: red;
font-weight: bold;
        }        tr:nth-
child(even) {
background-color: red;
        }
</style>
</head>
<h1>Tables</h1>
${tables.map(table => `
    <table>
      <thead>
        <tr>
          ${table.metaData.map(col => `<th>${col.name}</th>`).join('')}
        </tr>
      </thead>
      <tbody>
        ${table.tableHTML}
      </tbody>
    </table>
  `).join('')}
<form action="/logout" method="post">

```

```
        <button type="submit">logout</button> ` `);
        } catch (error) {
            console.error(error);
            res.send('Error Occurred');
        } finally {
        }
    });
    app.post('/logout', async (req, res) => {
        try {
            // await connection.close();
            const log = `[${new Date().toString()}] User ${req.body.username} logged
out\n`;
            fs.appendFile('logs.txt', log, (err) => {
                if (err) throw err;
                console.log('The log was saved!');
            });
        } catch (error) {
            console.error(error);
        }
        res.redirect('/');
    });

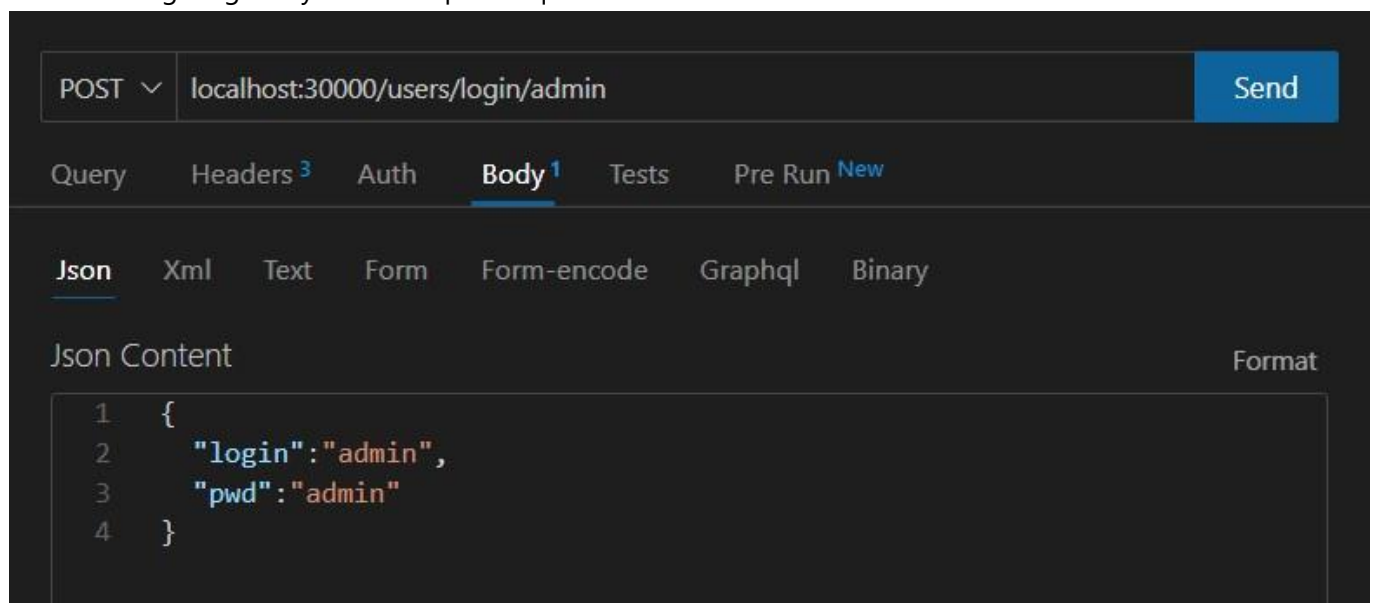
    app.listen(3000, () => console.log('Server Started'));
```

this code only shows the tables as a result of the login either its admin or simple user

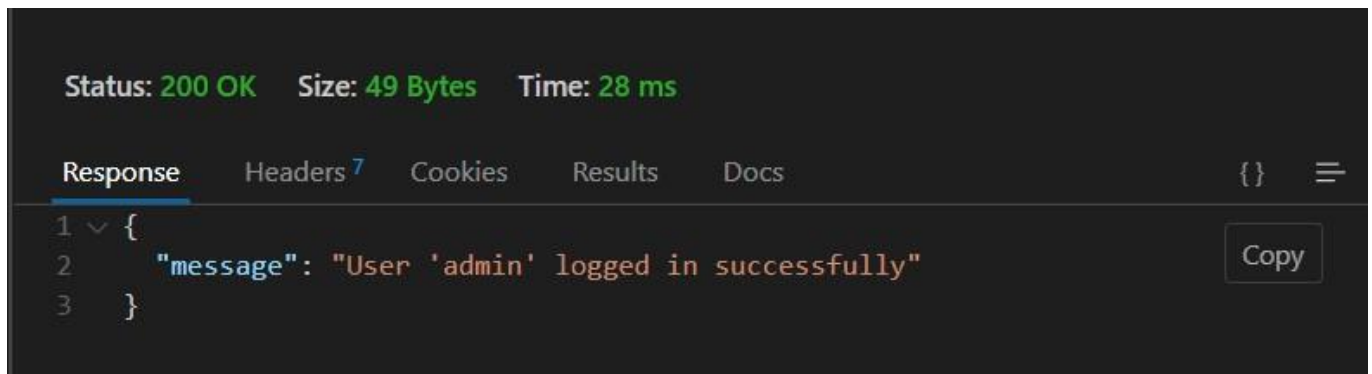
Test using Thunder client :

connection as an admin :

Now we are going to try to send a post request to connect as an admin



here are the results:



This is a Node.js code using the Express framework to handle an HTTP POST request to the '/login' endpoint. The code receives a username and password in the request body. It then checks if a user with the provided username and password exists in an array of allowed users. If a user is found, the code attempts to establish a connection to an Oracle database using the provided username and password. If the connection is successful, it sends a success message back to the client with a 200 status code. If the user is not found, it sends a failure message with a 500 status code.

```
app.post('/login', async (req, res) => {

  const { login, pwd } = req.body;

  const user = allowedUsers.find(u => u.username === login && u.password === pwd);

  if (!user) {

    return res.status(500).send({ message: 'Login Failed, user not valid' });

  }

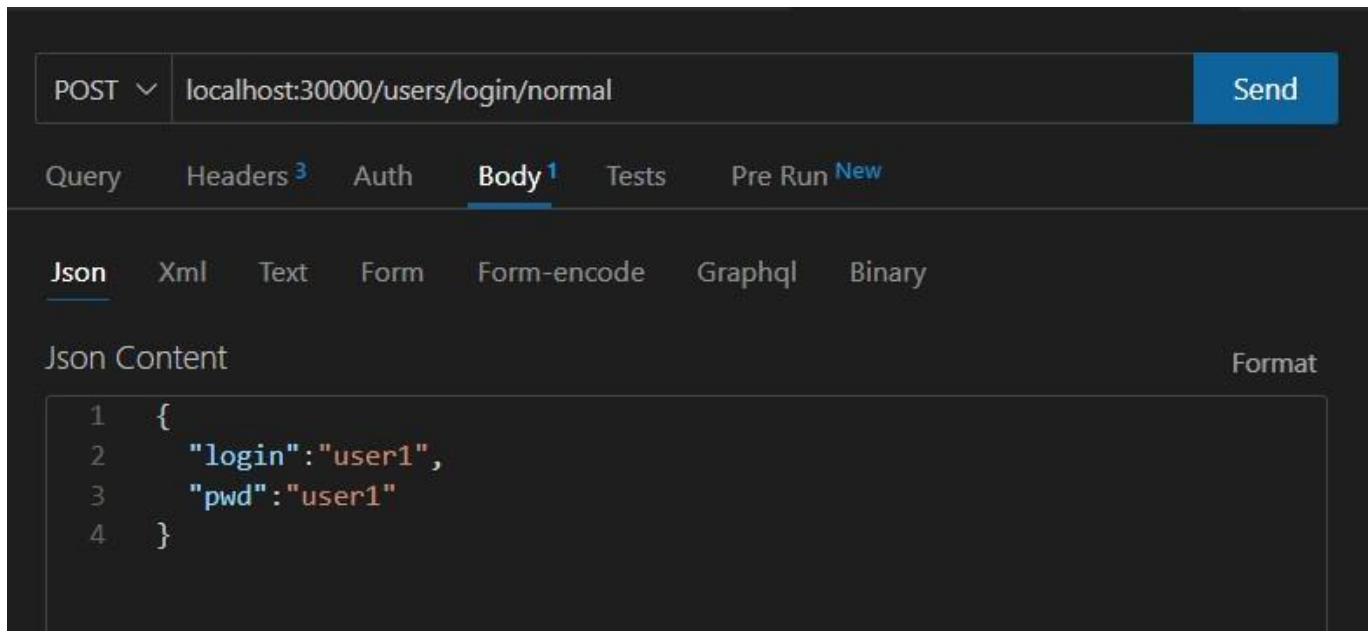
  try {

    const connection = await oracledb.getConnection({
      user: login,
      password: pwd,
      connectString: "localhost/orc1"
    })

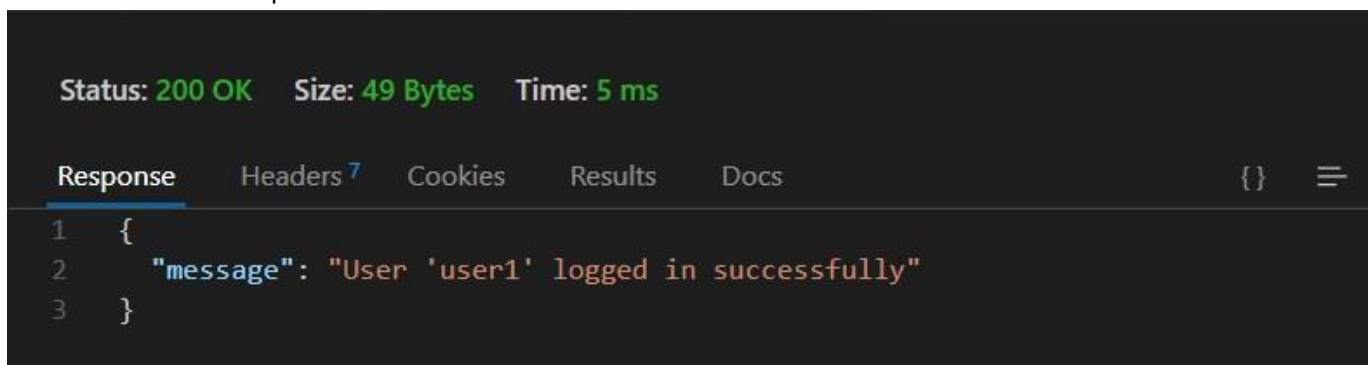
    if(connection)
      return res.status(200).send({message: "User "+login+" logged in successfully"})
    }catch (err) {
      res.status(404).json({ message: "auth failed" });
    }
  }
```

connection as anormal user :

now lets sent a Post request in order to log as a noraml user



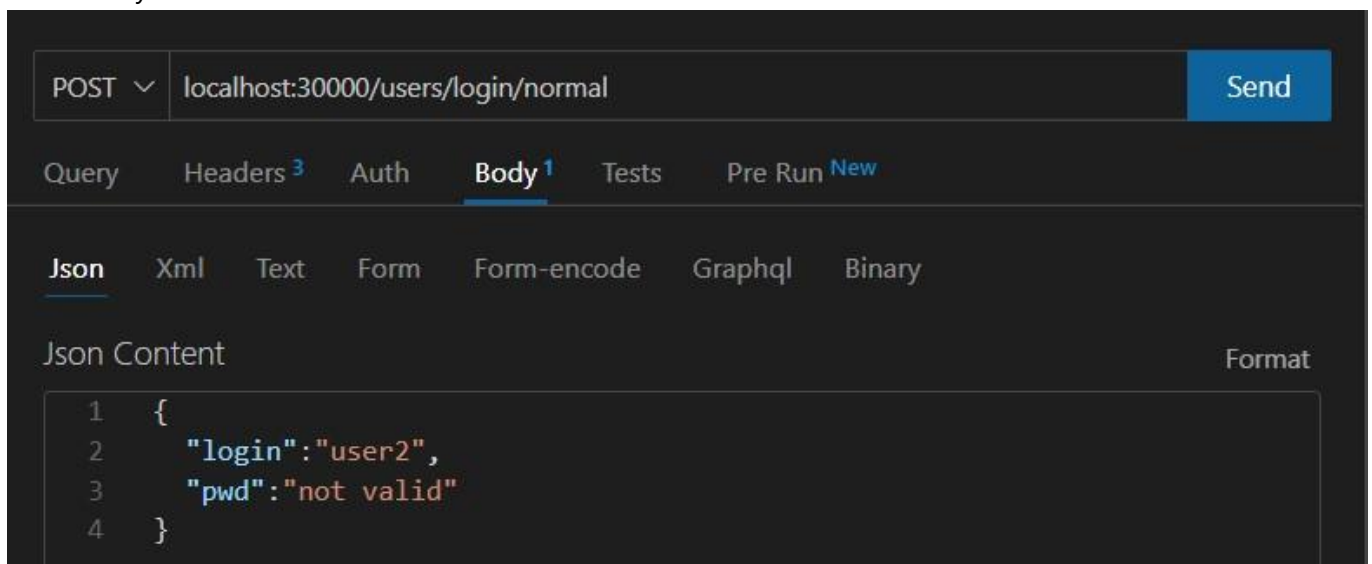
lets see the server response :



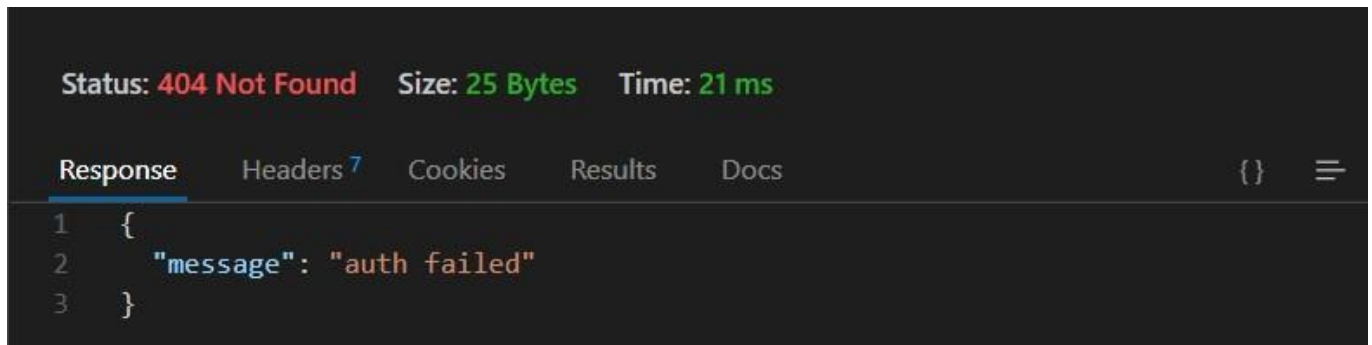
The code tries to connect to an Oracle database using the provided identifying information and checks to see if the user name and password provided in the body of the request correspond to an authorized user in the list allowedUsers.

Failed connection :

now lets try a user that do not exist



we recieve a 404 error msg saying that the user does not exist



getting marks :

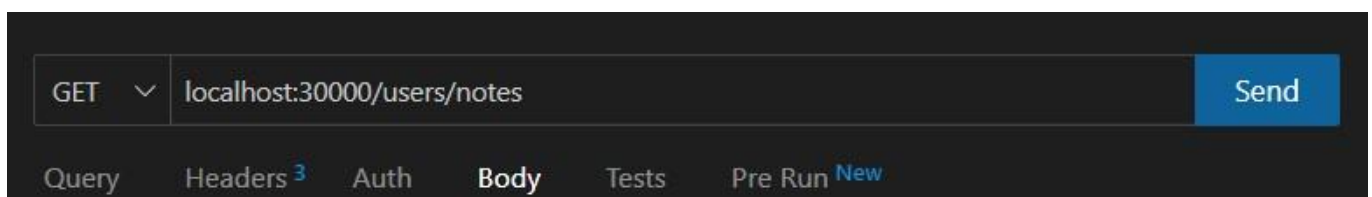
first we logging as a student and using the jwt process it keeps us logged as user1 to do that we simply add a middleware in our app.js

```
app.use((req, res, next) => {
  try {
    const token = req.headers.authorization.split(' ')[1];
    const decoded = jwt.verify(token, process.env.JWT_SECRET);
    req.userData = decoded;
    next();
  } catch (error) {
    return res.status(401).json({
      message: 'Auth failed'
    });
  }
});
```

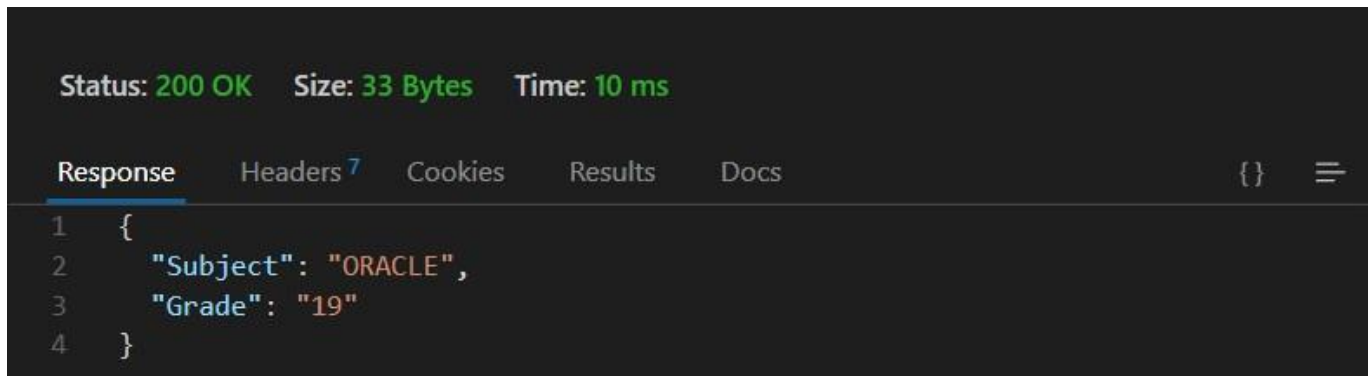
When a request is made to the application, this middleware is executed first. It tries to extract a JSON Web Token (JWT) from the "Authorization" header of the request. The token is then decrypted using the "jwt.verify()" method and the JWT secret stored in the environment variable "process.env.JWT_SECRET".

If the decryption is successful, the decoded token is added to the request object as a property "req.userData", and the control is passed to the next middleware or route handler using "next()".

If there is any error in the process, for example, the token is missing or invalid, the middleware returns a JSON response with a status code of 401 (Unauthorized) and a message "Auth failed". now lets send a request to retrieve data :



the response is the following :



This is a code snippet in Node.js that sets up a connection to an Oracle database using the `oracledb` library. The code establishes a connection to the database using the user and password specified, and the connection string (`localhost/orc1`).

The code then sets up an endpoint in the Express app using the `app.get` method. This endpoint retrieves student marks from the `projet.notes` table in the Oracle database by executing an SQL statement. The result of the query is returned as a JSON object to the client.

```
oracledb.getConnection({  
  user: req.session.login,  
  password: password, connectionString: 'localhost/orc1'  
}, (err, connection) => {  
  if (err) {  
    console.error(err);  
    return;  
  }  
  
  app.get('/allmarks', (req, res) => {  
    sql = 'SELECT * FROM projet.notes';  
    connection.execute(sql, (err, result) => {  
      return res.json (result.rows);  
    });  
  });  
});
```

but we can't modify using `user1` because it does not have the privilege

```
Status: 500 Internal Server Error   Size: 61 Bytes   Time: 11 ms

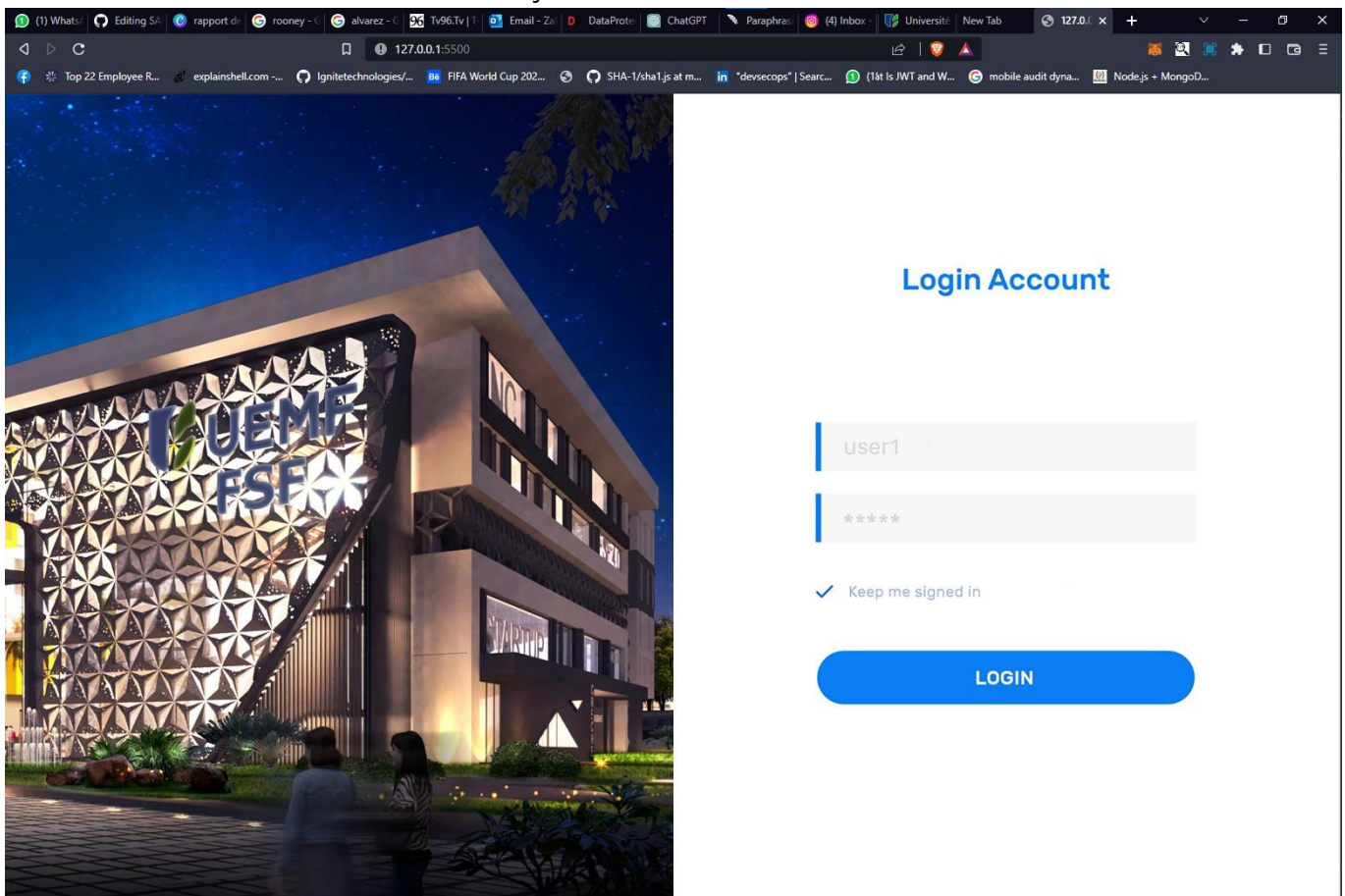
Response   Headers 7   Cookies   Results   Docs

1  {
2    "messgae": "this user does not have the privilege to modify"
3  }
```

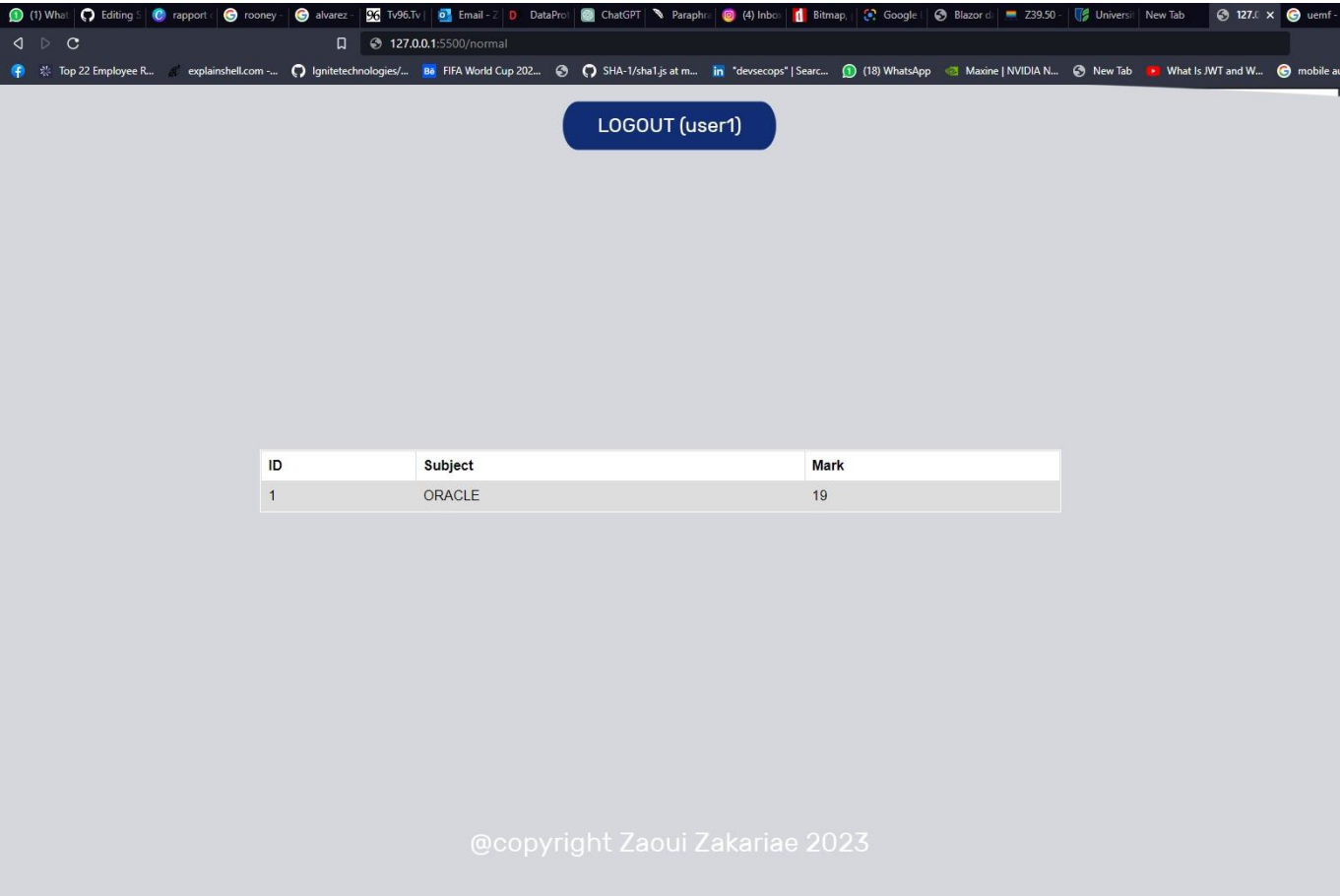
we have to change the logging to admin in order to do this action

Web platform test:

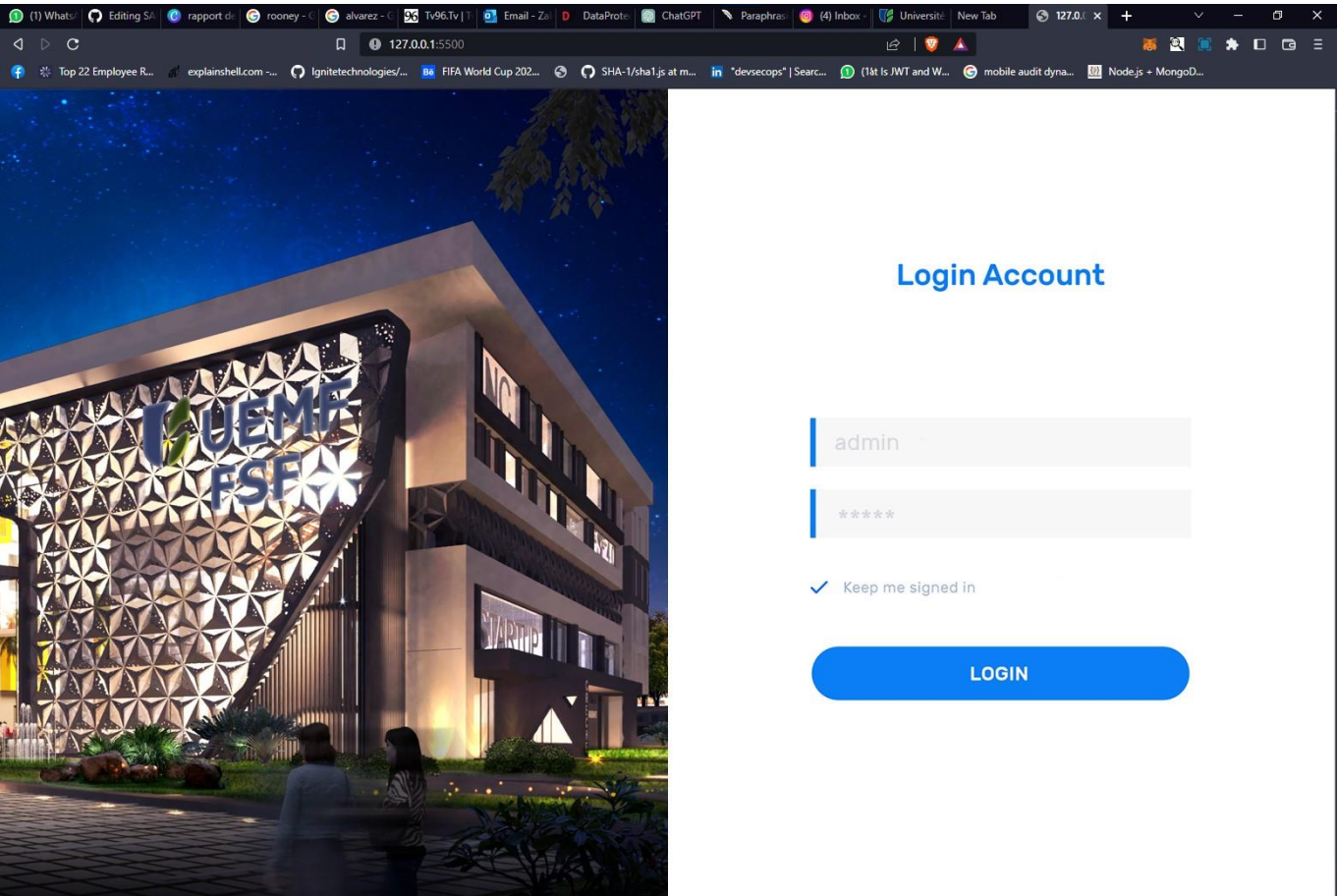
The code checks to see if the connection information corresponds to one of the authorized users when the user logs in by entering their username and password in a form on the home page. If the information is accurate, the user will be directed to a page appropriate for their user type (admin or normal). Additionally, the code records the connection's time in a journal file.



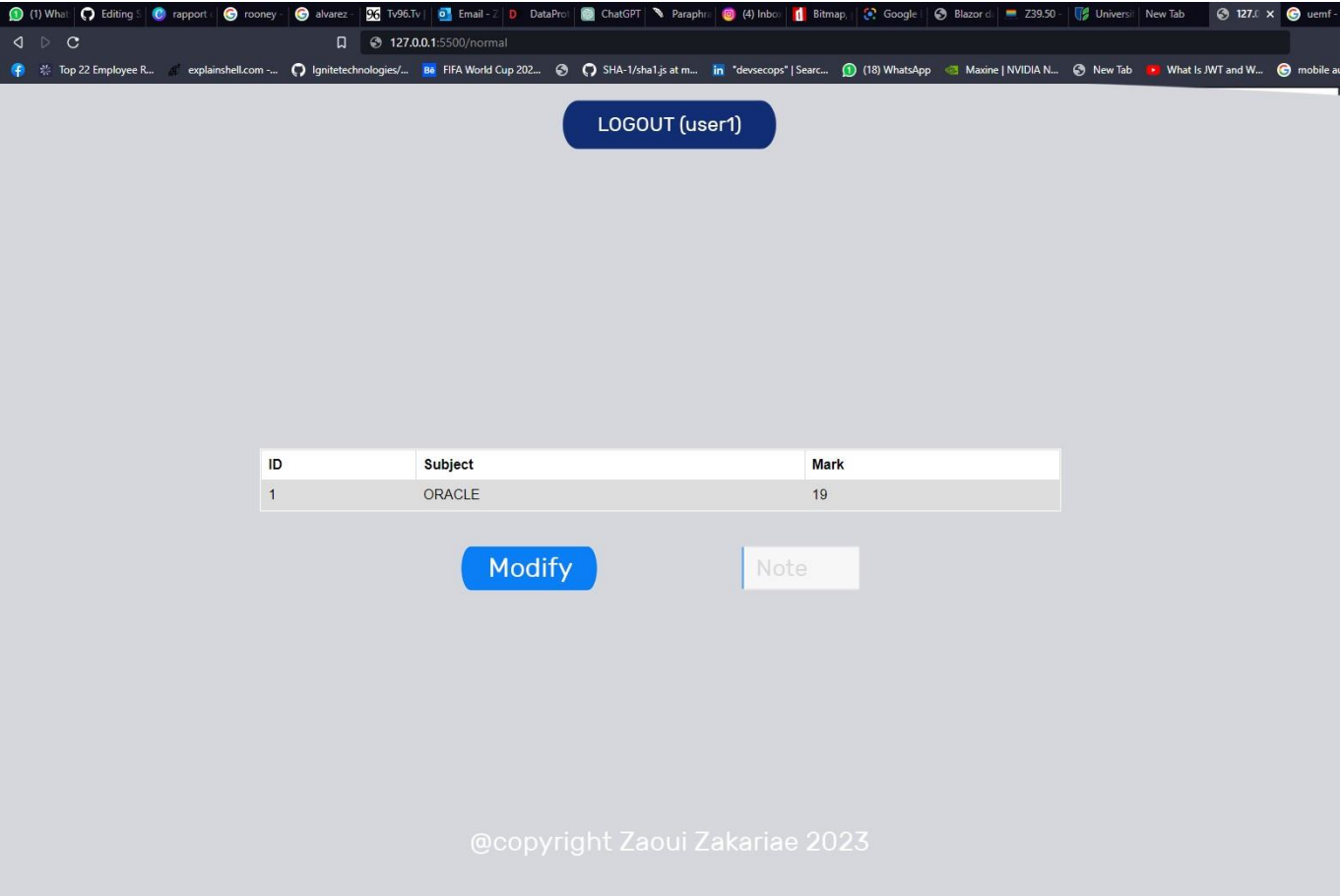
if the user is a normal user like user1 he can only view the table of marks :



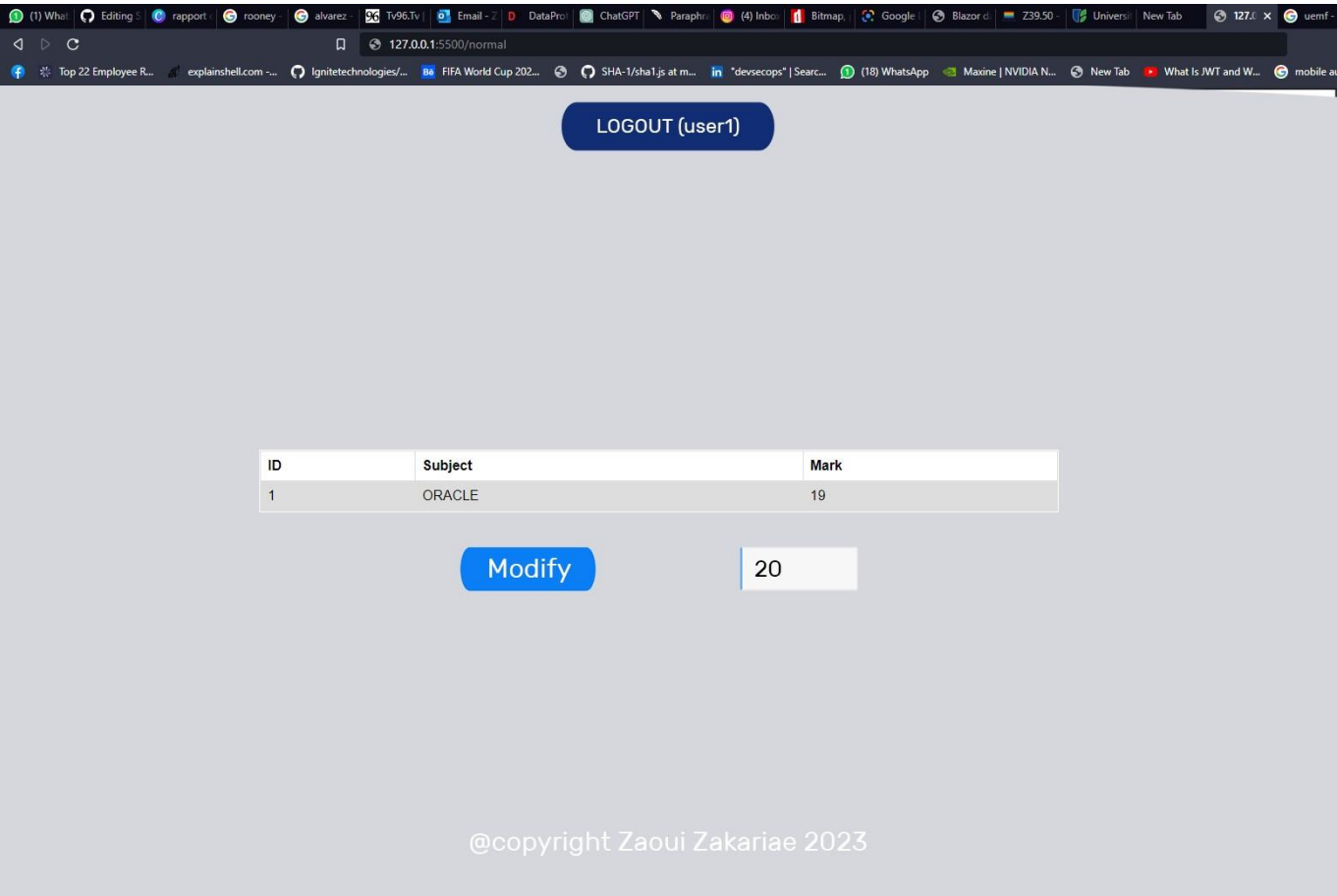
If the user is an administrator, they can view a page displaying the database's table of marks and he can modify on it. The code uses the OracleDB library to run some SQL queries to modify the data in this table and generate an HTML table for each result. Here is an example of a connection with admin:



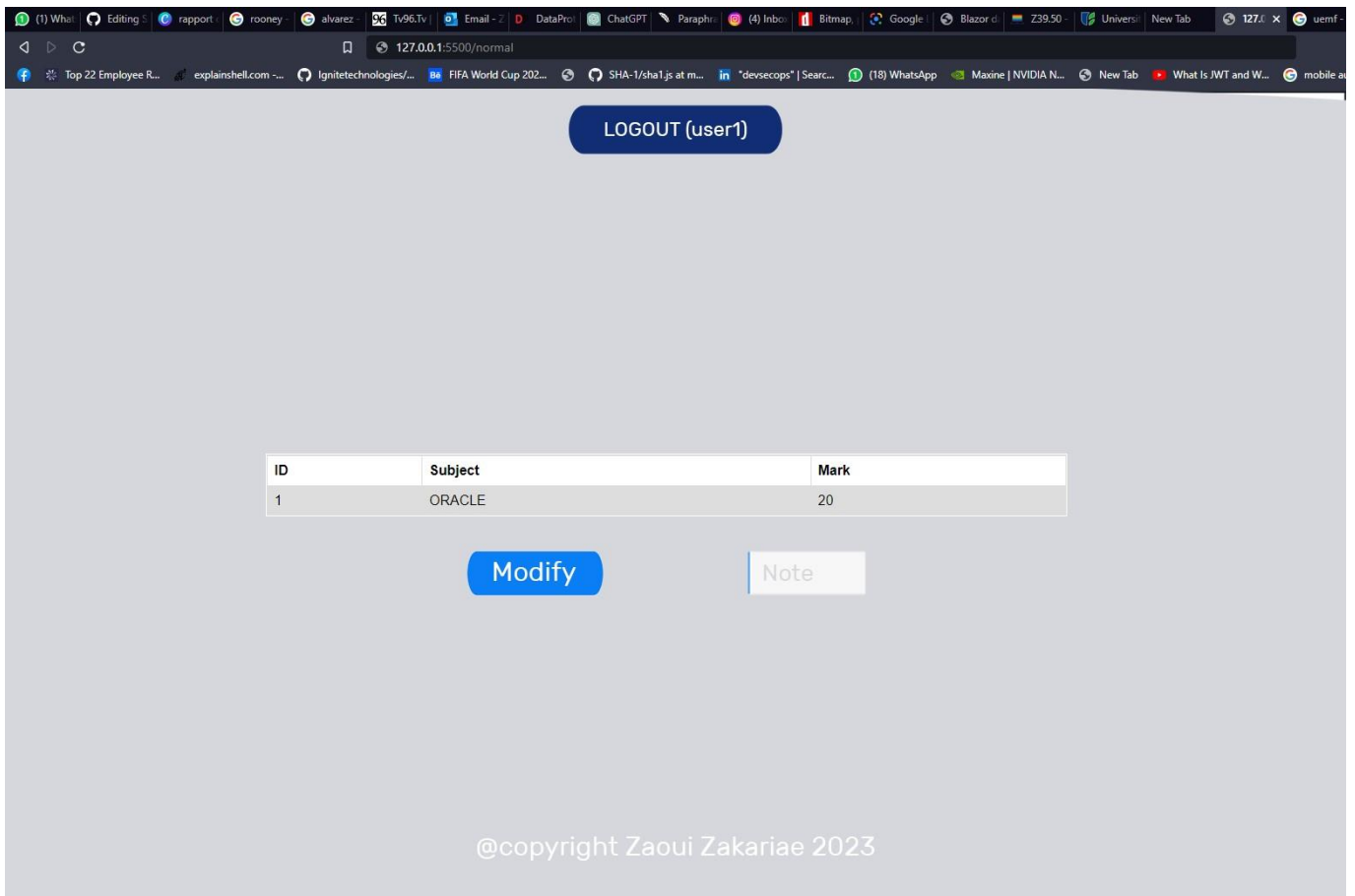
- first we are going to login as an administrator using the id "admin" and the password "admin" :



- as you can see we got a web page containing the table in addition of a button and an input to modify the mark



- Here we change the mark to 20



- after clicking "modify" button we can see the table is updated

we can access the file logs.txt to see our journal , and the logs of all the users with details

```
[Wed Feb 01 2023 01:20:29 GMT+0100 (UTC+02:00)] User user1 logged in
[Wed Feb 01 2023 01:20:31 GMT+0100 (UTC+02:00)] User user1 logged out [Wed
Feb 01 2023 01:20:51 GMT+0100 (UTC+02:00)] User admin logged in
[Wed Feb 01 2023 01:21:05 GMT+0100 (UTC+02:00)] User admin logged out
```

Conclusion :

In summary, this project gave us a thorough education in database design and development using Oracle software. By building a database system with features such as a login system and displaying tables, we gained practical insights into real-world database applications. The project sharpened our skills in creating, managing and querying databases, and gave us a deeper understanding of their significance in modern information systems. We found the project to be valuable and enlightening, and feel confident in tackling similar projects in the future.

End ...