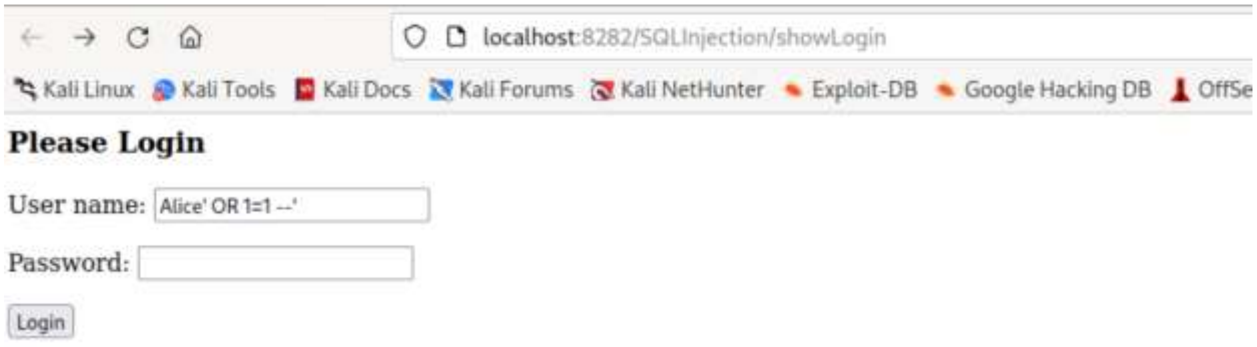
Task 5.2C: SQL Injection

- To complete this task, I decided to look into the SQL Injection web application since it is vulnerable to the SQL injection attacks. It was to look for the loop hole in the code which was creating the problem and then exploit that loop hole in order to login without the proper password.

Making Use of the Vulnerability

I entered "Alice' OR 1=1 --"

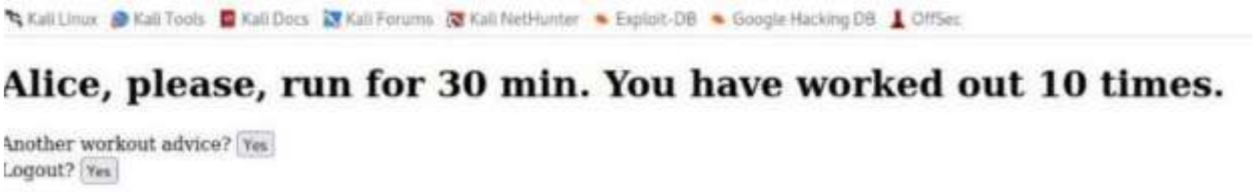

localhost:8282/SQLInjection/showLogin

Please Login

User name: Alice' OR 1=1 --'

Password:

- With this SQL injection string I obtained the access to the application even though I typed wrong password. The application flaw is not very hard to spot since it inserts the username field directly into the SQL query without proper validation.

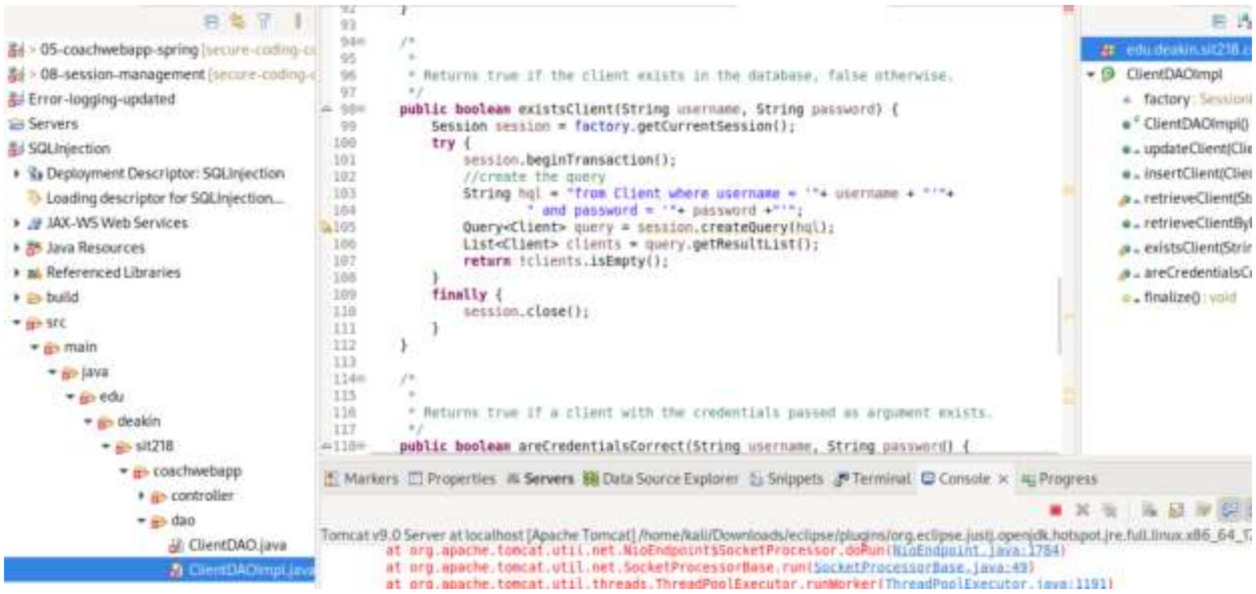

Alice, please, run for 30 min. You have worked out 10 times.

Another workout advice? Yes

Logout? Yes

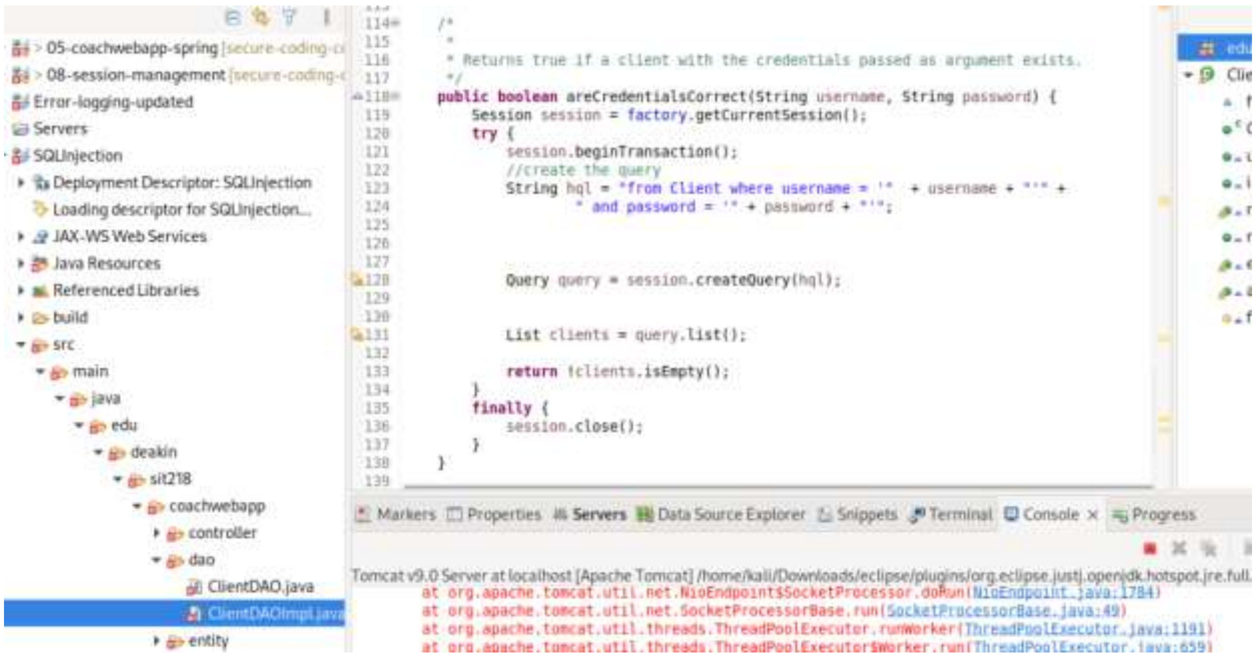
How to Determine the Vulnerable Code

- There are well set protocols. The client has the vulnerable code, and the credentials are correct. The issue is that the SQL query becomes joined immediately with the input login and password. Due to this, the system is exposed to cases of SQL injection attacks because of poor sanitization on the input data.



```
    }

    /*
     *
     * Returns true if the client exists in the database, false otherwise.
     */
    public boolean existsClient(String username, String password) {
        Session session = factory.getCurrentSession();
        try {
            session.beginTransaction();
            //create the query
            String hql = "from Client where username = '"+ username + "'"+
                    " and password = '"+ password +"'";
            Query<Client> query = session.createQuery(hql);
            List<Client> clients = query.getResultList();
            return !clients.isEmpty();
        }
        finally {
            session.close();
        }
    }

    /*
     *
     * Returns true if a client with the credentials passed as argument exists.
     */
    public boolean areCredentialsCorrect(String username, String password) {
```

```
Tomcat v9.0 Server at localhost [Apache Tomcat] /home/kali/Downloads/eclipse/plugins/org.eclipse.justj.openjdk.hotspot.jre.full.linux.x86_64_1
        at org.apache.tomcat.util.net.NioEndpoint$SocketProcessor.doRun(NioEndpoint.java:1784)
        at org.apache.tomcat.util.net.SocketProcessorBase.run(SocketProcessorBase.java:49)
        at org.apache.tomcat.util.threads.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1191)
```



```
    /*
     *
     * Returns true if a client with the credentials passed as argument exists.
     */
    public boolean areCredentialsCorrect(String username, String password) {
        Session session = factory.getCurrentSession();
        try {
            session.beginTransaction();
            //create the query
            String hql = "from Client where username = '"  + username + "'" +
                    " and password = '" + password + "'";



            Query query = session.createQuery(hql);


            List clients = query.list();

            return !clients.isEmpty();
        }
        finally {
            session.close();
        }
    }
```

```
Tomcat v9.0 Server at localhost [Apache Tomcat] /home/kali/Downloads/eclipse/plugins/org.eclipse.justj.openjdk.hotspot.jre.full.
        at org.apache.tomcat.util.net.NioEndpoint$SocketProcessor.doRun(NioEndpoint.java:1784)
        at org.apache.tomcat.util.net.SocketProcessorBase.run(SocketProcessorBase.java:49)
        at org.apache.tomcat.util.threads.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1191)
        at org.apache.tomcat.util.threads.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:659)
```

This code's weakness is caused by

- Insufficient Input Validation The program inserts their input into the query without the basic evaluation of the potential threats. Straightforward SQL Concatenation since a SQL query is constructed from strings, an attacker can easily alter the query's mechanics.

Reasons for Code Vulnerability

- The issue is that it's wording of the question itself. This is because an attacker is able to bypass the authentication step by including malicious SQL code into the SQL statement such as (OR 1=1). This occurs due to SQL statement = TRUE allowing the attacker to get in without a password.

The correct piece of code. (You don't have to execute and show the output)

- The usernames and passwords directly typed by the users in the previous code were inserted directly into the SQL query. An application which utilizes such a mechanism is vulnerable to SQL injection attacks. For instance, any raw SQL instructions entered can be manipulated by an attacker to corrupt the database, but if he succeeds in changing the inputs, then he is granted unauthorized access.

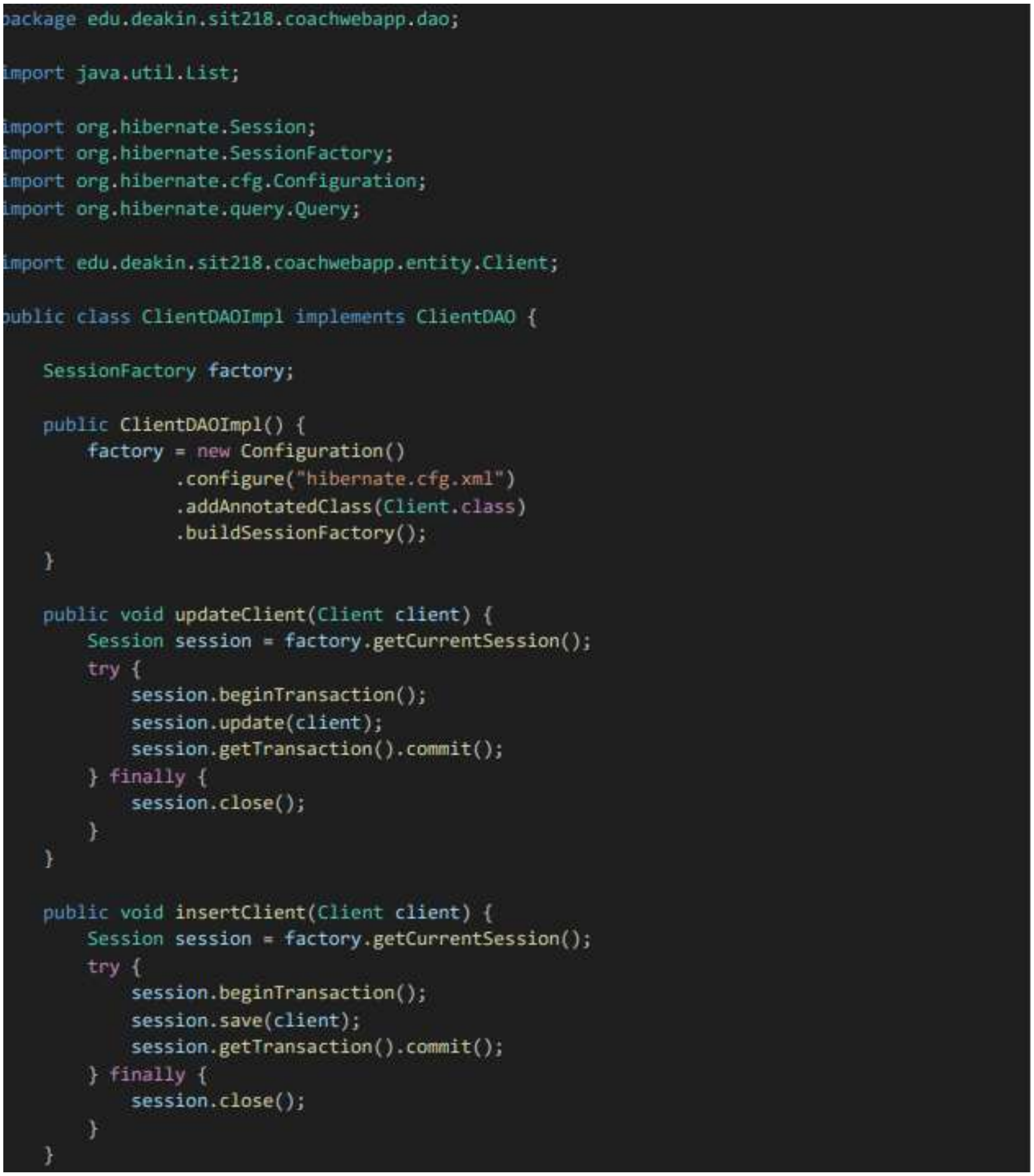
Queries with parameters

- In the created HQL (Hibernate Query Language) queries, the updated code uses named parameters. :USER and PASS are placeholders used here instead of using directly for the inputs gathered from the user.

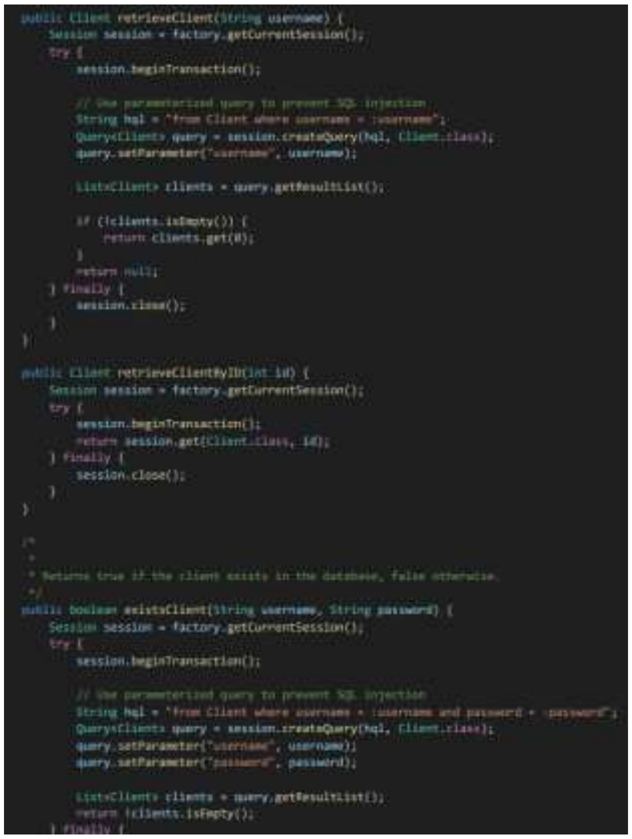
Safe Management of Input

- The setParameter() method is used in order to in as far as possible safely bind the actual user-input values to the set query parameters. This is a benefit as it adjoins attackers from entering special characters as well as other strings by enabling Hibernate to escape.

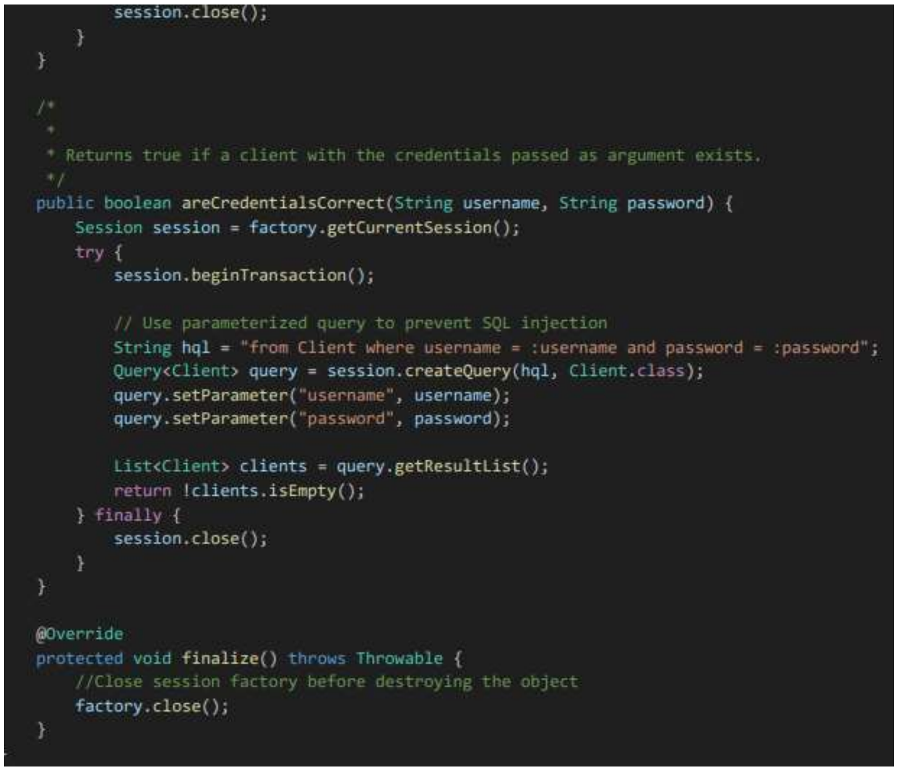
- The new code is more secure because parameterized queries are used, which means that any entered value by the user cannot be interpreted by the application as part of the command. This method also assist in making the applied system secure because the user inputs are not handled as code to reduce the chance of being targeted by the SQL injection attacks.



```
package edu.deakin.sit218.coachwebapp.dao;

import java.util.List;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;
import org.hibernate.query.Query;

import edu.deakin.sit218.coachwebapp.entity.Client;

public class ClientDAOImpl implements ClientDAO {

    SessionFactory factory;

    public ClientDAOImpl() {
        factory = new Configuration()
                .configure("hibernate.cfg.xml")
                .addAnnotatedClass(Client.class)
                .buildSessionFactory();
    }

    public void updateClient(Client client) {
        Session session = factory.getCurrentSession();
        try {
            session.beginTransaction();
            session.update(client);
            session.getTransaction().commit();
        } finally {
            session.close();
        }
    }

    public void insertClient(Client client) {
        Session session = factory.getCurrentSession();
        try {
            session.beginTransaction();
            session.save(client);
            session.getTransaction().commit();
        } finally {
            session.close();
        }
    }
```



```
public Client retrieveClient(String username) {
    Session session = factory.getCurrentSession();
    try {
        session.beginTransaction();

        // Use parameterized query to prevent SQL injection
        String hql = "from Client where username = :username";
        Query<Client> query = session.createQuery(hql, Client.class);
        query.setParameter("username", username);

        List<Client> clients = query.getResultList();

        if (!clients.isEmpty()) {
            return clients.get(0);
        }
        return null;
    } finally {
        session.close();
    }
}

public Client retrieveClientByID(int id) {
    Session session = factory.getCurrentSession();
    try {
        session.beginTransaction();
        return session.get(Client.class, id);
    } finally {
        session.close();
    }
}

/*
 *
 * Returns true if the client exists in the database, false otherwise.
 */
public boolean existsClient(String username, String password) {
    Session session = factory.getCurrentSession();
    try {
        session.beginTransaction();

        // Use parameterized query to prevent SQL injection
        String hql = "from Client where username = :username and password = :password";
        Query<Client> query = session.createQuery(hql, Client.class);
        query.setParameter("username", username);
        query.setParameter("password", password);

        List<Client> clients = query.getResultList();
        return !clients.isEmpty();
    } finally {
```



```
        session.close();
    }
}

/*
 *
 * Returns true if a client with the credentials passed as argument exists.
 */
public boolean areCredentialsCorrect(String username, String password) {
    Session session = factory.getCurrentSession();
    try {
        session.beginTransaction();

        // Use parameterized query to prevent SQL injection
        String hql = "from Client where username = :username and password = :password";
        Query<Client> query = session.createQuery(hql, Client.class);
        query.setParameter("username", username);
        query.setParameter("password", password);

        List<Client> clients = query.getResultList();
        return !clients.isEmpty();
    } finally {
        session.close();
    }
}

@Override
protected void finalize() throws Throwable {
    //Close session factory before destroying the object
    factory.close();
}
```