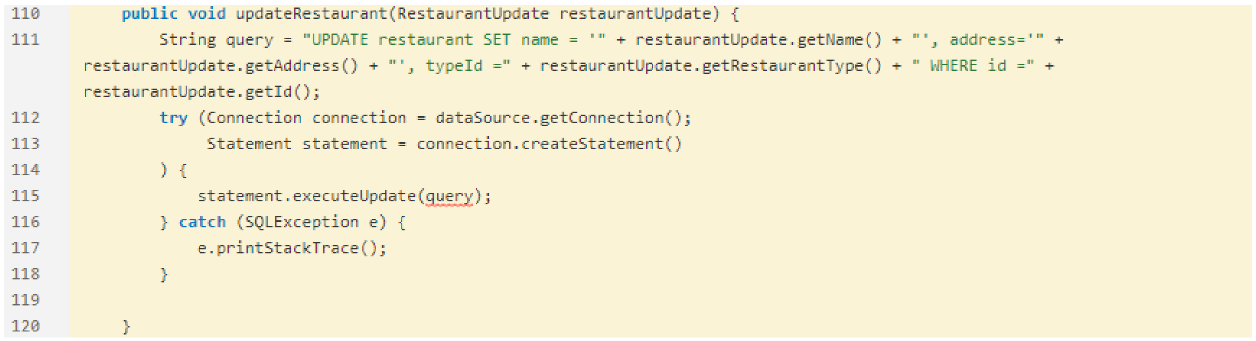
SonarQube izveštaj

Jovana Kitanović 3156/21

Security Hotspots: SQL Injection

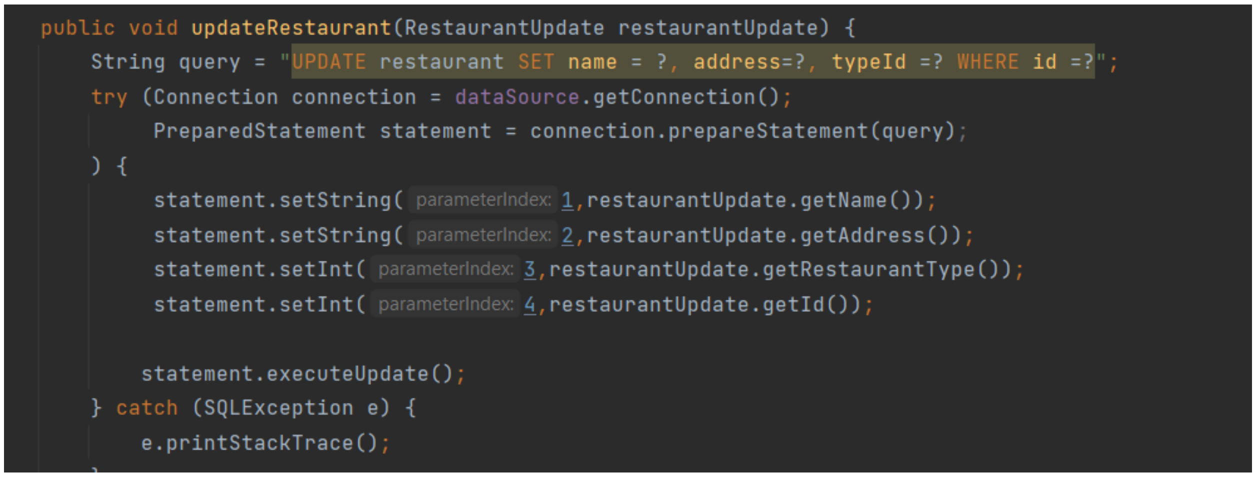
src/.../securesoftwaredevelopment/repository/CustomerRepository.java



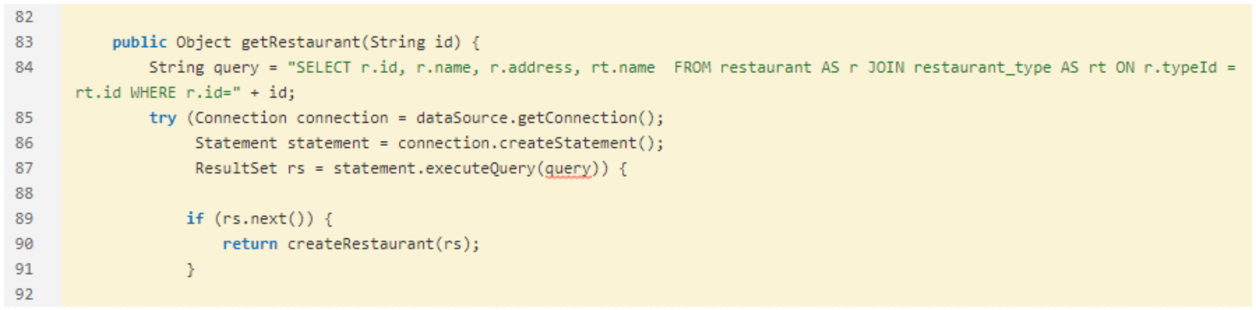
```
    public void updateRestaurant(RestaurantUpdate restaurantUpdate) {
        String query = "UPDATE restaurant SET name = '" + restaurantUpdate.getName() + "', address='" +
restaurantUpdate.getAddress() + "', typeId =" + restaurantUpdate.getRestaurantType() + " WHERE id =" +
restaurantUpdate.getId();
        try (Connection connection = dataSource.getConnection();
             Statement statement = connection.createStatement()
        ) {
            statement.executeUpdate(query);
        } catch (SQLException e) {
            e.printStackTrace();
        }

    }
```

Status: true positive

Funkcija se poziva prilikom potvrđivanja ažuriranih podataka o restoranu. Kako forma za unos ima polja za slobodan unos teksta, preko kojih je moguće izvršiti SQL injection napad, potrebno je promeniti funkciju tako da se upit izršava korišćenjem prepared statement-a



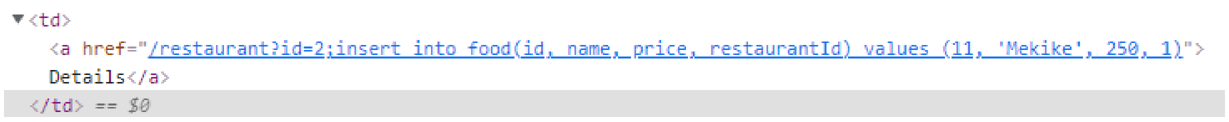
```
public void updateRestaurant(RestaurantUpdate restaurantUpdate) {
    String query = "UPDATE restaurant SET name = ?, address=?, typeId =? WHERE id =?";
    try (Connection connection = dataSource.getConnection();
         PreparedStatement statement = connection.prepareStatement(query);
    ) {
         statement.setString( parameterIndex: 1,restaurantUpdate.getName());
         statement.setString( parameterIndex: 2,restaurantUpdate.getAddress());
         statement.setInt( parameterIndex: 3,restaurantUpdate.getRestaurantType());
         statement.setInt( parameterIndex: 4,restaurantUpdate.getId());

        statement.executeUpdate();
    } catch (SQLException e) {
        e.printStackTrace();
```



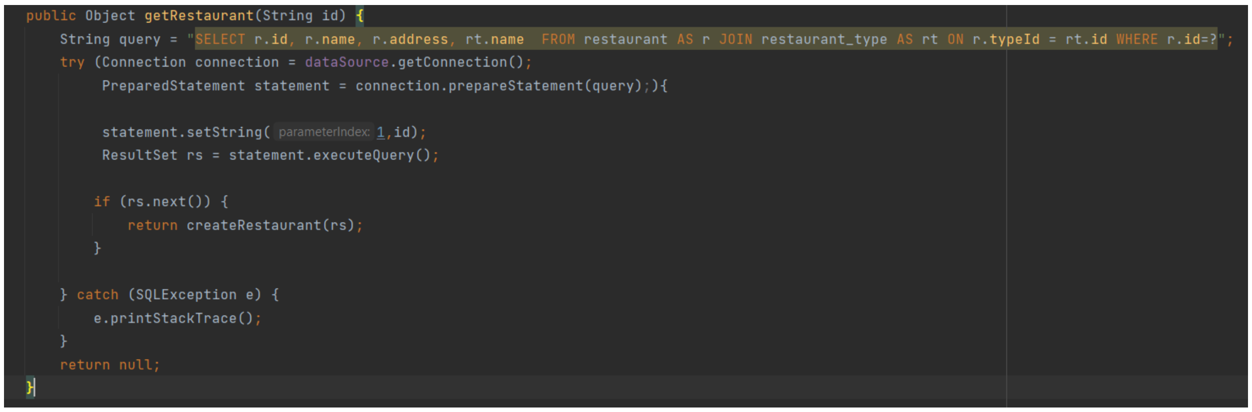
```

    public Object getRestaurant(String id) {
        String query = "SELECT r.id, r.name, r.address, rt.name  FROM restaurant AS r JOIN restaurant_type AS rt ON r.typeId =
rt.id WHERE r.id=" + id;
        try (Connection connection = dataSource.getConnection();
             Statement statement = connection.createStatement();
             ResultSet rs = statement.executeQuery(query)) {

            if (rs.next()) {
                return createRestaurant(rs);
            }

```

Status: true positive

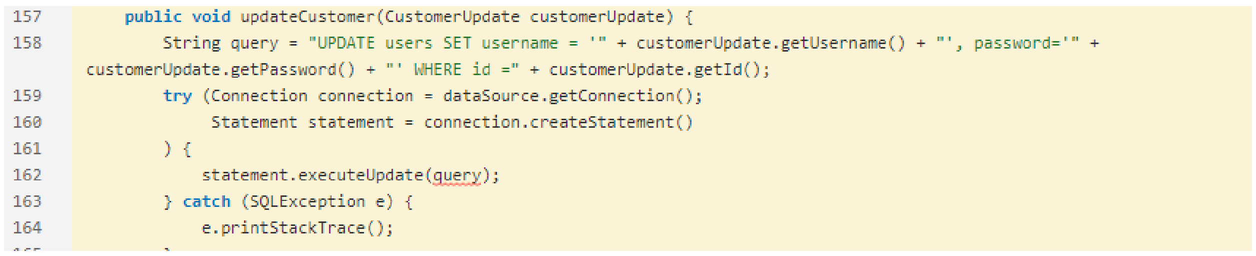
Potrebno je dodati zaštitu od SQL injection-a, jer zlonamerni korisnik, dodavanjem SQL upita na link ka pregledu restorana može uspešno izvršiti napad. Kao na slici ispod pasusa, ukoliko bi korisnik kliknuo na details opciju restorana dva u bazu hrane bi se dodalo novo jelo.



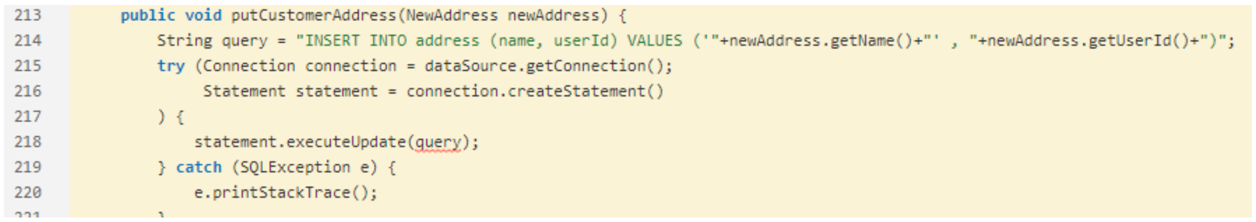
```
▼<td>
   <a href="/restaurant?id=2;insert into food(id, name, price, restaurantId) values (11, 'Mekike', 250, 1)">
   Details</a>
 </td> == $0
```



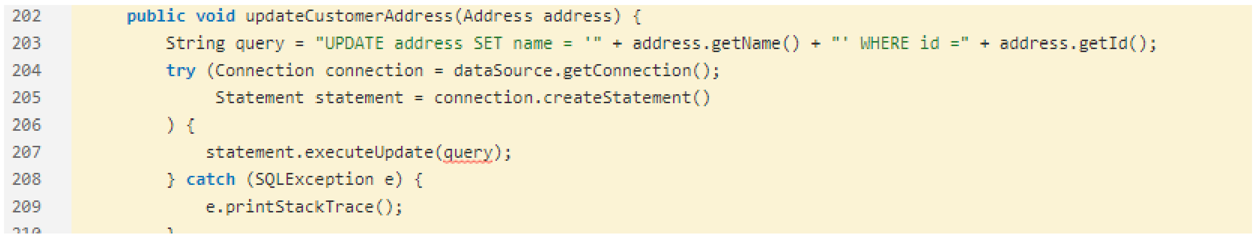
```
public Object getRestaurant(String id) {
    String query = "SELECT r.id, r.name, r.address, rt.name  FROM restaurant AS r JOIN restaurant_type AS rt ON r.typeId = rt.id WHERE r.id=?";
    try (Connection connection = dataSource.getConnection();
         PreparedStatement statement = connection.prepareStatement(query);){

         statement.setString( parameterIndex: 1,id);
         ResultSet rs = statement.executeQuery();

        if (rs.next()) {
            return createRestaurant(rs);
        }

    } catch (SQLException e) {
        e.printStackTrace();
    }
    return null;
}
```



```
    public void updateCustomer(CustomerUpdate customerUpdate) {
        String query = "UPDATE users SET username = '" + customerUpdate.getUsername() + "', password='" +
customerUpdate.getPassword() + "' WHERE id =" + customerUpdate.getId();
        try (Connection connection = dataSource.getConnection();
             Statement statement = connection.createStatement()
        ) {
            statement.executeUpdate(query);
        } catch (SQLException e) {
            e.printStackTrace();
```



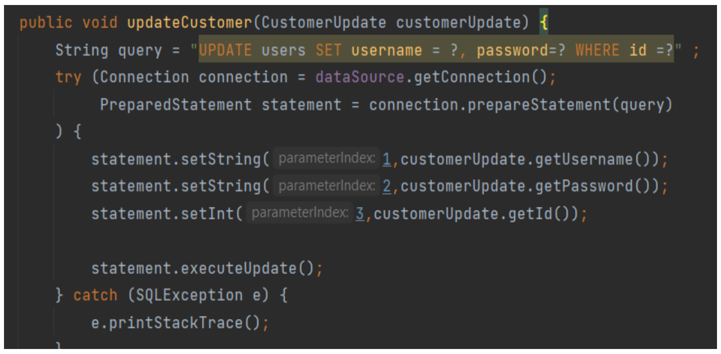
```
    public void putCustomerAddress(NewAddress newAddress) {
        String query = "INSERT INTO address (name, userId) VALUES ('"+newAddress.getName()+"' , "+newAddress.getUserId()+")";
        try (Connection connection = dataSource.getConnection();
             Statement statement = connection.createStatement()
        ) {
            statement.executeUpdate(query);
        } catch (SQLException e) {
            e.printStackTrace();
```



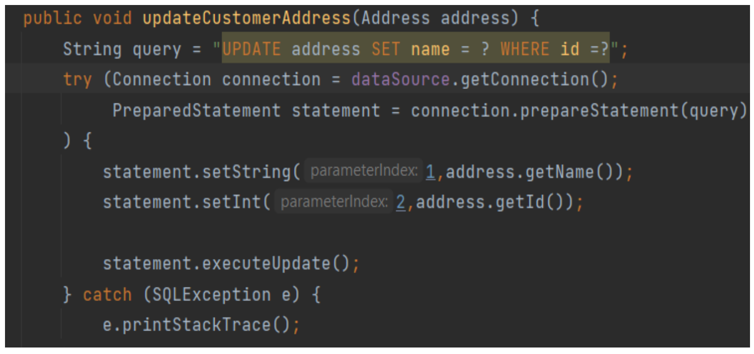
```
    public void updateCustomerAddress(Address address) {
        String query = "UPDATE address SET name = '" + address.getName() + "' WHERE id =" + address.getId();
        try (Connection connection = dataSource.getConnection();
             Statement statement = connection.createStatement()
        ) {
            statement.executeUpdate(query);
        } catch (SQLException e) {
            e.printStackTrace();
```

Status: true positive

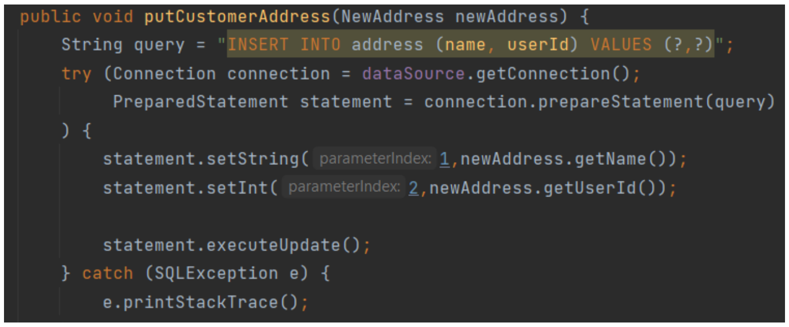
Funkcija update customer se poziva prilikom potvrđivanja ažuriranih podataka o kupcu. Kako forma za unos ima polja za slobodan unos teksta, preko kojih je moguće izvršiti SQL injection napad, potrebno je promeniti funkciju tako da se upit izršava korišćenjem prepared statement-a. Ista situacija je i sa uređivanjem adrese korisnika i dodavanjem nove adrese.

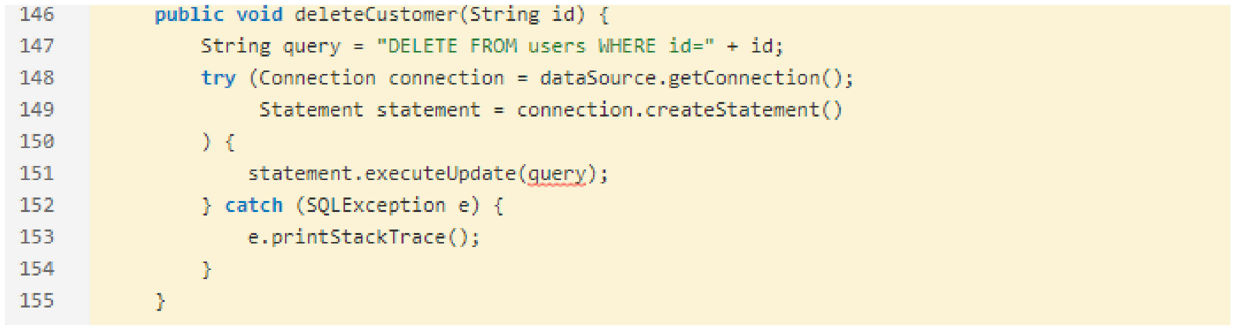


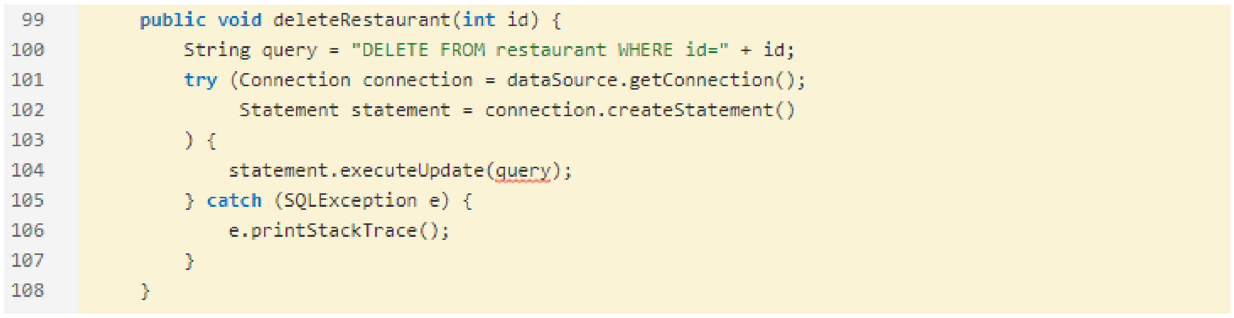
```
public void updateCustomer(CustomerUpdate customerUpdate) {
    String query = "UPDATE users SET username = ?, password=? WHERE id =?" ;
    try (Connection connection = dataSource.getConnection();
         PreparedStatement statement = connection.prepareStatement(query)
    ) {
        statement.setString( parameterIndex: 1,customerUpdate.getUsername());
        statement.setString( parameterIndex: 2,customerUpdate.getPassword());
        statement.setInt( parameterIndex: 3,customerUpdate.getId());

        statement.executeUpdate();
    } catch (SQLException e) {
        e.printStackTrace();
```

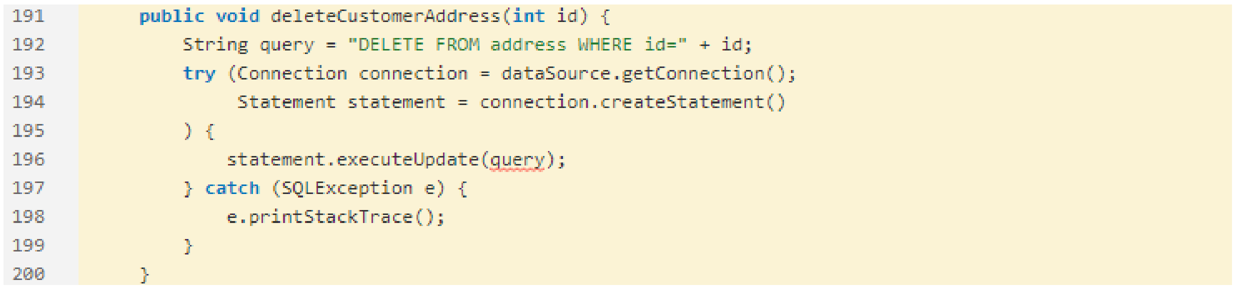


```
public void updateCustomerAddress(Address address) {
    String query = "UPDATE address SET name = ? WHERE id =?";
    try (Connection connection = dataSource.getConnection();
         PreparedStatement statement = connection.prepareStatement(query)
    ) {
        statement.setString( parameterIndex: 1,address.getName());
        statement.setInt( parameterIndex: 2,address.getId());

        statement.executeUpdate();
    } catch (SQLException e) {
        e.printStackTrace();
```


```java
public void putCustomerAddress(NewAddress newAddress) {
    String query = "INSERT INTO address (name, userId) VALUES (?,?)";
    try (Connection connection = dataSource.getConnection();
         PreparedStatement statement = connection.prepareStatement(query)
    ) {
        statement.setString( parameterIndex: 1,newAddress.getName());
        statement.setInt( parameterIndex: 2,newAddress.getUserId());

        statement.executeUpdate();
    } catch (SQLException e) {
        e.printStackTrace();
```

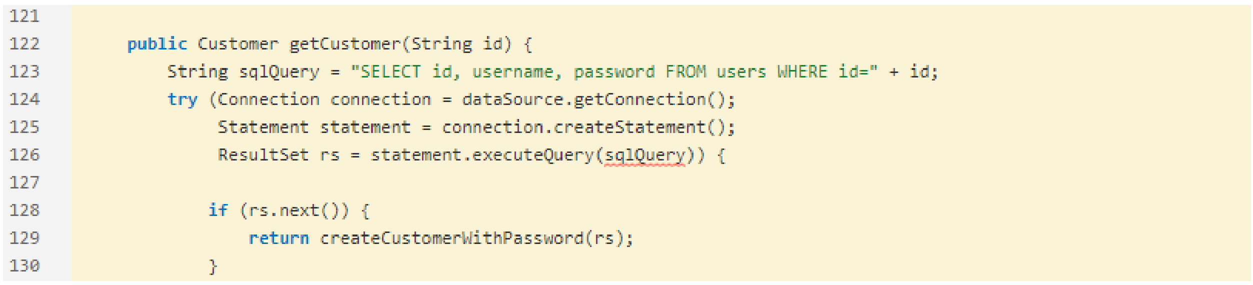

```java
    public void deleteCustomer(String id) {
        String query = "DELETE FROM users WHERE id=" + id;
        try (Connection connection = dataSource.getConnection();
             Statement statement = connection.createStatement()
        ) {
            statement.executeUpdate(query);
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
```

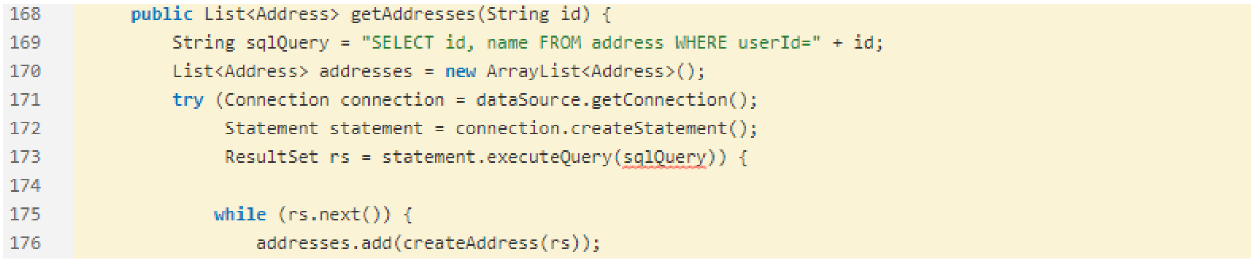

```java
    public void deleteRestaurant(int id) {
        String query = "DELETE FROM restaurant WHERE id=" + id;
        try (Connection connection = dataSource.getConnection();
             Statement statement = connection.createStatement()
        ) {
            statement.executeUpdate(query);
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
```


```java
    public void deleteCustomerAddress(int id) {
        String query = "DELETE FROM address WHERE id=" + id;
        try (Connection connection = dataSource.getConnection();
             Statement statement = connection.createStatement()
        ) {
            statement.executeUpdate(query);
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
```

Status: false positive

Funkcijom i upitom se ne može izvršit SQL injection napad.

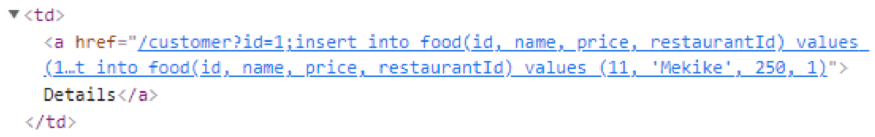

```java

    public Customer getCustomer(String id) {
        String sqlQuery = "SELECT id, username, password FROM users WHERE id=" + id;
        try (Connection connection = dataSource.getConnection();
             Statement statement = connection.createStatement();
             ResultSet rs = statement.executeQuery(sqlQuery)) {

            if (rs.next()) {
                return createCustomerWithPassword(rs);
            }
```



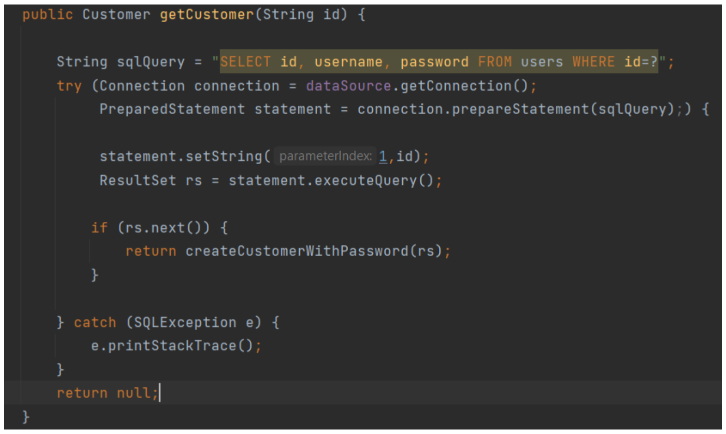
```java
    public List<Address> getAddresses(String id) {
        String sqlQuery = "SELECT id, name FROM address WHERE userId=" + id;
        List<Address> addresses = new ArrayList<Address>();
        try (Connection connection = dataSource.getConnection();
             Statement statement = connection.createStatement();
             ResultSet rs = statement.executeQuery(sqlQuery)) {

            while (rs.next()) {
                addresses.add(createAddress(rs));
```

Status: true positive

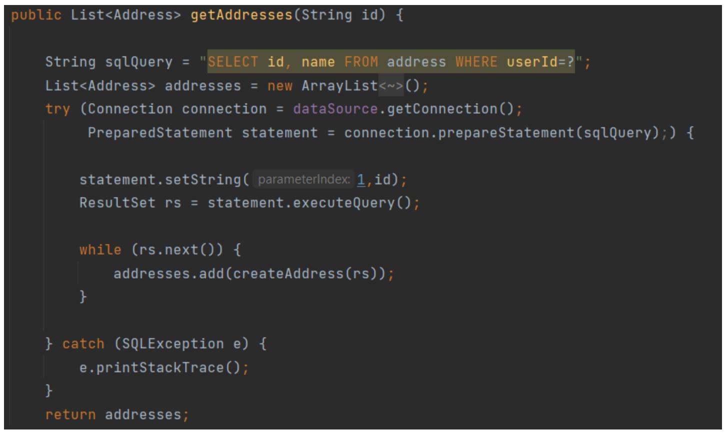
Funkcije getCustomer I getAddress se pozivaju jedna za drugom kada korisnik želi da pogleda zapis nekog od korisnika i obe mogu izazvati SQL napad. Naime, funkcije kao parametar prihvataju iz url-a identifinator korisnika koji treba da bude tekstualnog tipa, ukoliko bi zlonamerni korisnik dodao upit u nastavku url-a, on bi se i izvršio.



```html
▼<td>
   <a href="/customer?id=1;insert into food(id, name, price, restaurantId) values (1…t into food(id, name, price, restaurantId) values (11, 'Mekike', 250, 1)">
   Details</a>
 </td>
```

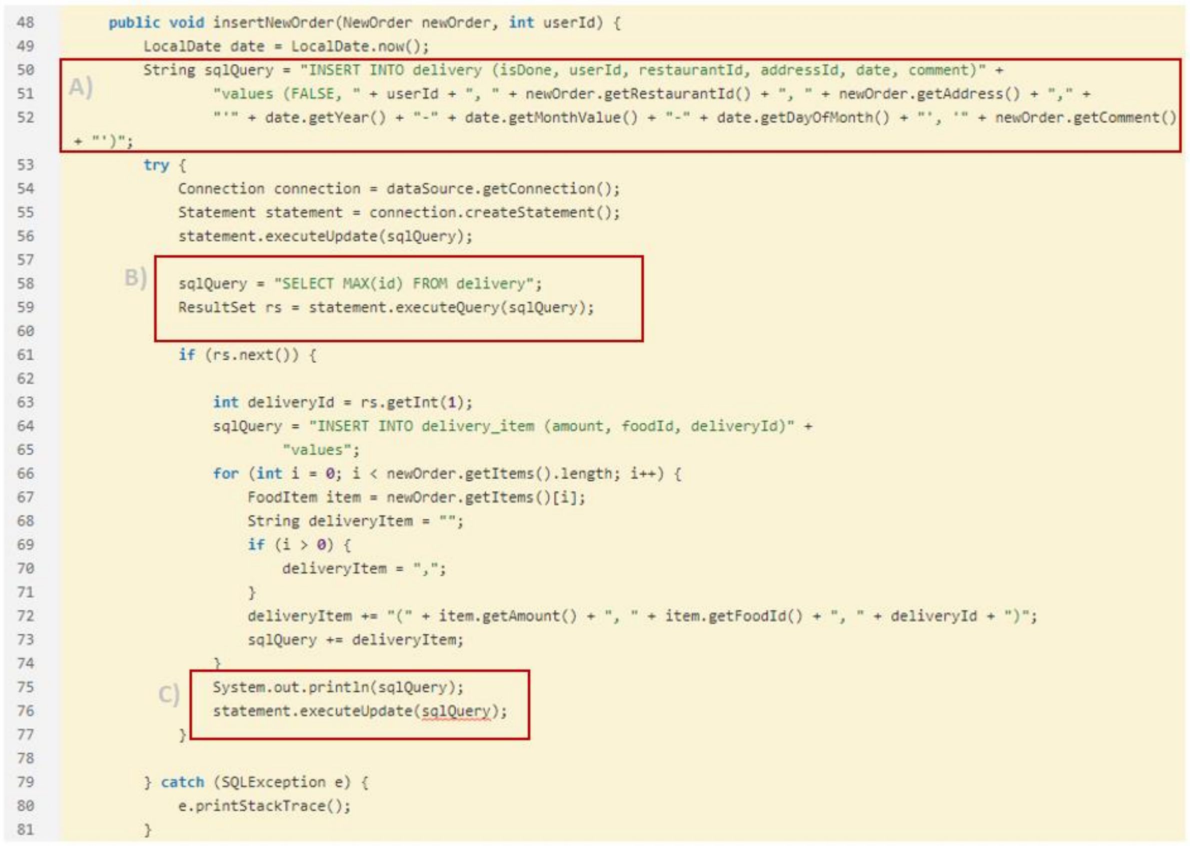


```java
public Customer getCustomer(String id) {

    String sqlQuery = "SELECT id, username, password FROM users WHERE id=?";
    try (Connection connection = dataSource.getConnection();
         PreparedStatement statement = connection.prepareStatement(sqlQuery);) {

         statement.setString( parameterIndex: 1,id);
         ResultSet rs = statement.executeQuery();

        if (rs.next()) {
            return createCustomerWithPassword(rs);
        }

    } catch (SQLException e) {
        e.printStackTrace();
    }
    return null;
}
```



```java
public List<Address> getAddresses(String id) {

    String sqlQuery = "SELECT id, name FROM address WHERE userId=?";
    List<Address> addresses = new ArrayList<~>();
    try (Connection connection = dataSource.getConnection();
         PreparedStatement statement = connection.prepareStatement(sqlQuery);) {

        statement.setString( parameterIndex: 1,id);
        ResultSet rs = statement.executeQuery();

        while (rs.next()) {
            addresses.add(createAddress(rs));
        }

    } catch (SQLException e) {
        e.printStackTrace();
    }
    return addresses;
```

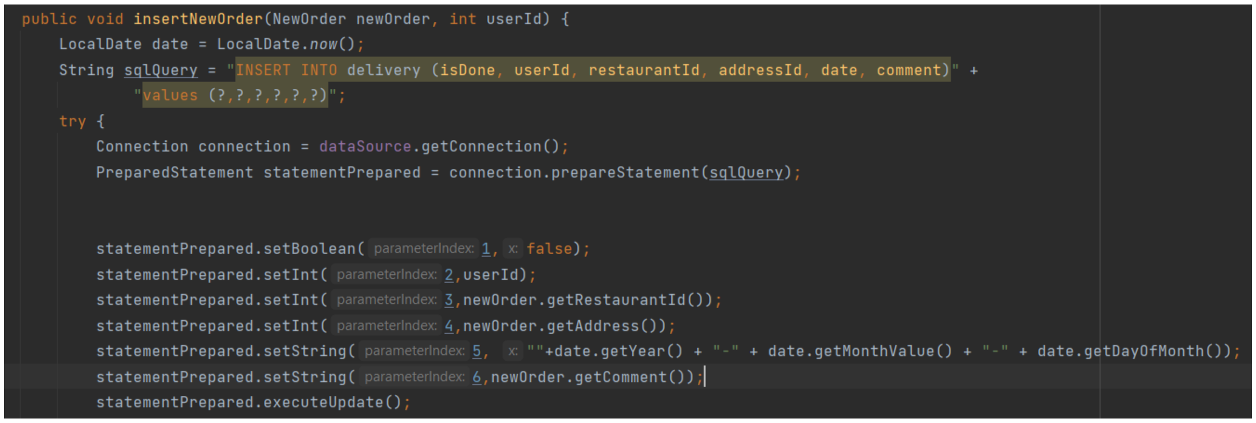
src/.../securesoftwaredevelopment/repository/OrderRepository.java



```java
    public void insertNewOrder(NewOrder newOrder, int userId) {
        LocalDate date = LocalDate.now();
        String sqlQuery = "INSERT INTO delivery (isDone, userId, restaurantId, addressId, date, comment)" +
                "values (FALSE, " + userId + ", " + newOrder.getRestaurantId() + ", " + newOrder.getAddress() + "," +
                "'" + date.getYear() + "-" + date.getMonthValue() + "-" + date.getDayOfMonth() + "', '" + newOrder.getComment()
+ "')";
        try {
            Connection connection = dataSource.getConnection();
            Statement statement = connection.createStatement();
            statement.executeUpdate(sqlQuery);

            sqlQuery = "SELECT MAX(id) FROM delivery";
            ResultSet rs = statement.executeQuery(sqlQuery);

            if (rs.next()) {

                int deliveryId = rs.getInt(1);
                sqlQuery = "INSERT INTO delivery_item (amount, foodId, deliveryId)" +
                        "values";
                for (int i = 0; i < newOrder.getItems().length; i++) {
                    FoodItem item = newOrder.getItems()[i];
                    String deliveryItem = "";
                    if (i > 0) {
                        deliveryItem = ",";
                    }
                    deliveryItem += "(" + item.getAmount() + ", " + item.getFoodId() + ", " + deliveryId + ")";
                    sqlQuery += deliveryItem;
                }
                System.out.println(sqlQuery);
                statement.executeUpdate(sqlQuery);
            }

        } catch (SQLException e) {
            e.printStackTrace();
        }
```

A) Status: true positive

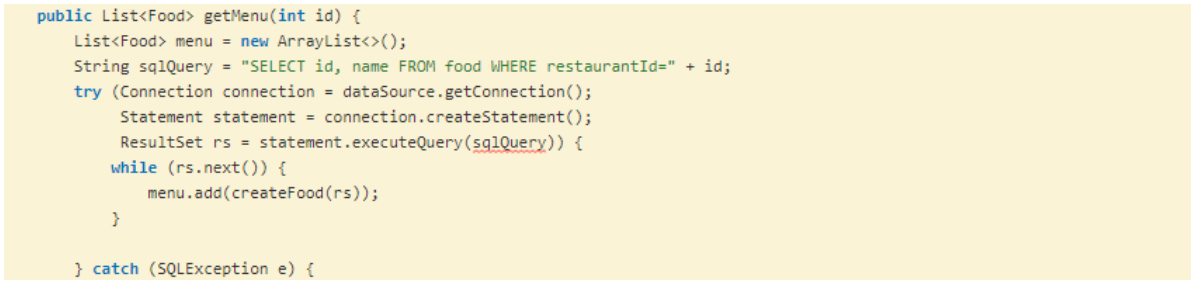
Funkcija se poziva prilikom dodavanja komentara uz porudžbinu, kako ne koristi preparedStatement, moguće je na ovom mestu izvršiti SQL napad. Rešenje problema je preuređivanje upita, tako da se parametri ne dodaju na upit kao slobodan teks, već da se kroz odgovarajuće funkcije prosleđuju prepared statement-u upita.



```java
public void insertNewOrder(NewOrder newOrder, int userId) {
    LocalDate date = LocalDate.now();
    String sqlQuery = "INSERT INTO delivery (isDone, userId, restaurantId, addressId, date, comment)" +
            "values (?,?,?,?,?,?)";
    try {
        Connection connection = dataSource.getConnection();
        PreparedStatement statementPrepared = connection.prepareStatement(sqlQuery);


        statementPrepared.setBoolean( parameterIndex: 1, x: false);
        statementPrepared.setInt( parameterIndex: 2,userId);
        statementPrepared.setInt( parameterIndex: 3,newOrder.getRestaurantId());
        statementPrepared.setInt( parameterIndex: 4,newOrder.getAddress());
        statementPrepared.setString( parameterIndex: 5, x: ""+date.getYear() + "-" + date.getMonthValue() + "-" + date.getDayOfMonth());
        statementPrepared.setString( parameterIndex: 6,newOrder.getComment());
        statementPrepared.executeUpdate();
```

B) i C) Status: false positive

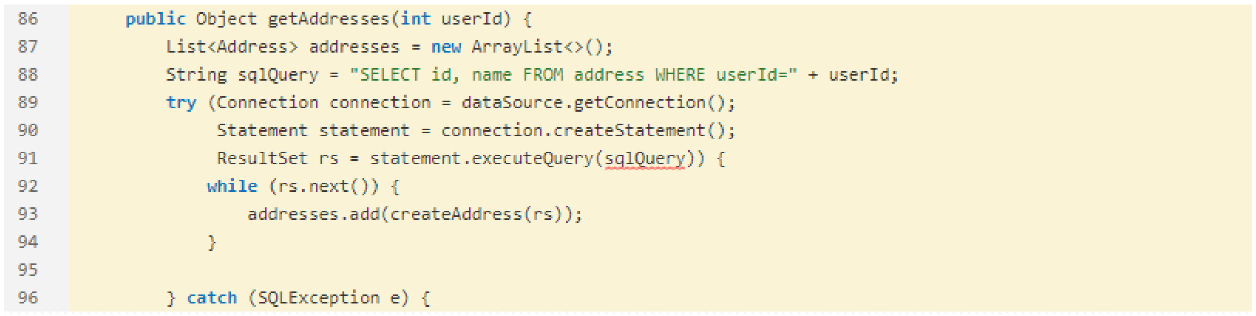
Upit pripada funkciji za dodavanje nove porudžbine. Kako se ni jedan od parametara koje korisnik sam unosi ne koristi u upitu, sql injection se ne može izvršiti.



```
public List<Food> getMenu(int id) {
    List<Food> menu = new ArrayList<>();
    String sqlQuery = "SELECT id, name FROM food WHERE restaurantId=" + id;
    try (Connection connection = dataSource.getConnection();
         Statement statement = connection.createStatement();
         ResultSet rs = statement.executeQuery(sqlQuery)) {
        while (rs.next()) {
            menu.add(createFood(rs));
        }

    } catch (SQLException e) {
```

Status: false positive

Funkcija se poziva kada se iz opadajućeg menija odabere neki od restorana. Pretnja je false positive jer sama funkcija prihvata samo integer, ukoliko bi napadač pokušao da proslei sql upit umesto id-ja restorana, bio bi podignut NumberFormatException

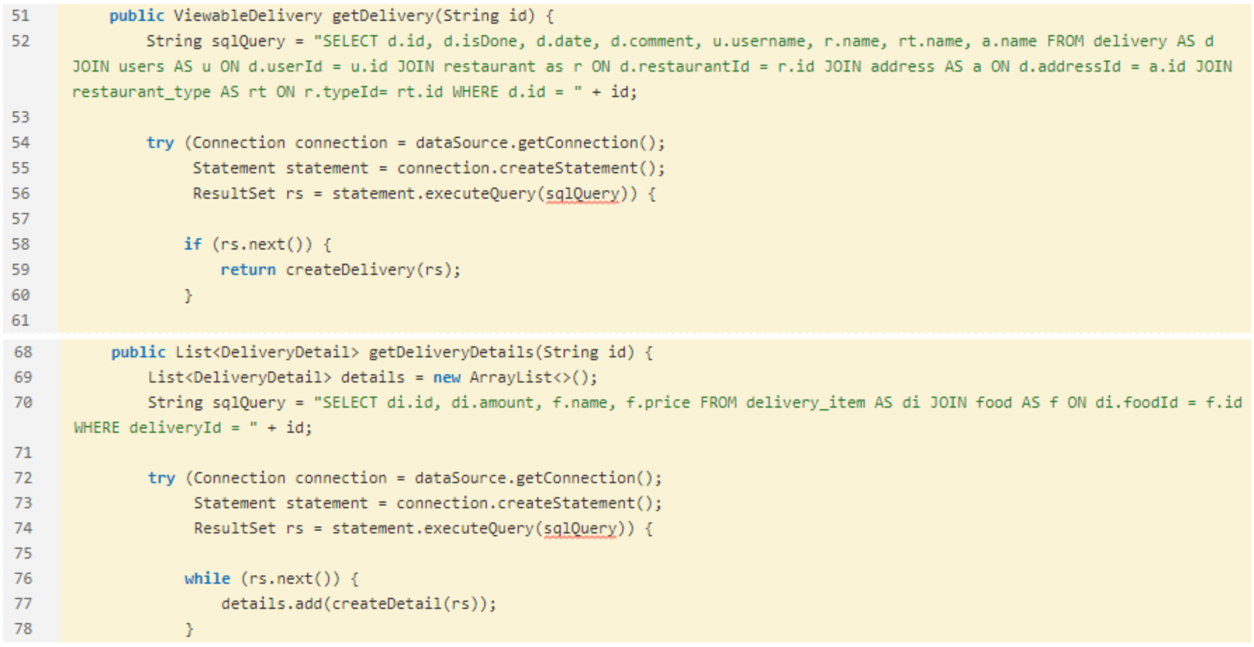


```
    public Object getAddresses(int userId) {
        List<Address> addresses = new ArrayList<>();
        String sqlQuery = "SELECT id, name FROM address WHERE userId=" + userId;
        try (Connection connection = dataSource.getConnection();
             Statement statement = connection.createStatement();
             ResultSet rs = statement.executeQuery(sqlQuery)) {
            while (rs.next()) {
                addresses.add(createAddress(rs));
            }

        } catch (SQLException e) {
```

Status: false positive

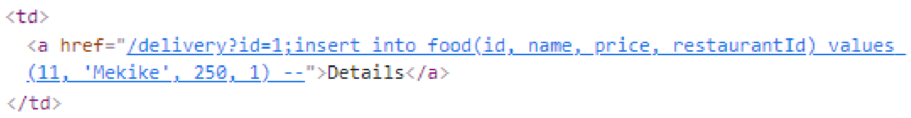
Funkcija se poziva kada ulogovani korisnik odabere opciju dodavanja nove porudžbine, kada ona za tog korisnika dohvata njegove adrese. Funkcija očekuje integer, tako da, ukoliko bi joj se prosledio string bila bi prijavljena greška.
Takođe, ukoliko se pokuša promena identifikatora adrese u inspect page odeljku, funkcija za dodavanje nove porudžbine bi prijavila gdešku jer umesto teksta očekuje broj.

src/.../securesoftwaredevelopment/repository/DeliveryRepository.java



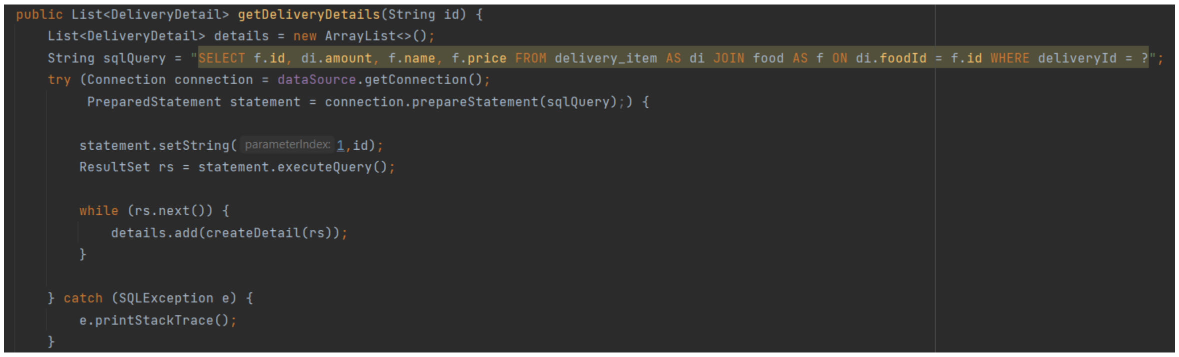
```java
    public ViewableDelivery getDelivery(String id) {
        String sqlQuery = "SELECT d.id, d.isDone, d.date, d.comment, u.username, r.name, rt.name, a.name FROM delivery AS d JOIN users AS u ON d.userId = u.id JOIN restaurant as r ON d.restaurantId = r.id JOIN address AS a ON d.addressId = a.id JOIN restaurant_type AS rt ON r.typeId= rt.id WHERE d.id = " + id;

        try (Connection connection = dataSource.getConnection();
             Statement statement = connection.createStatement();
             ResultSet rs = statement.executeQuery(sqlQuery)) {

            if (rs.next()) {
                return createDelivery(rs);
            }

    public List<DeliveryDetail> getDeliveryDetails(String id) {
        List<DeliveryDetail> details = new ArrayList<>();
        String sqlQuery = "SELECT di.id, di.amount, f.name, f.price FROM delivery_item AS di JOIN food AS f ON di.foodId = f.id WHERE deliveryId = " + id;

        try (Connection connection = dataSource.getConnection();
             Statement statement = connection.createStatement();
             ResultSet rs = statement.executeQuery(sqlQuery)) {

            while (rs.next()) {
                details.add(createDetail(rs));
            }
```

Status: true positive

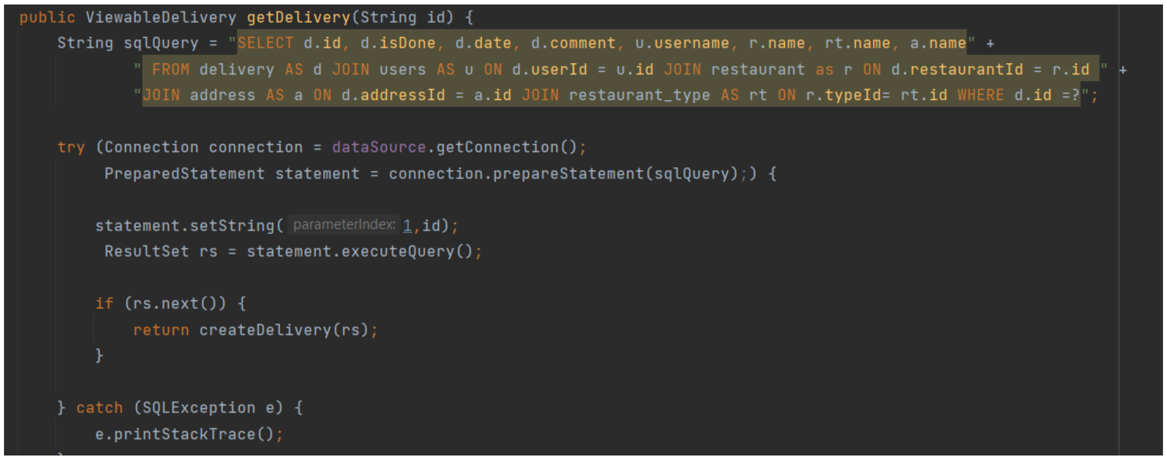


```html
<td>
  <a href="/delivery?id=1;insert into food(id, name, price, restaurantId) values (11, 'Mekike', 250, 1) --">Details</a>
</td>
```

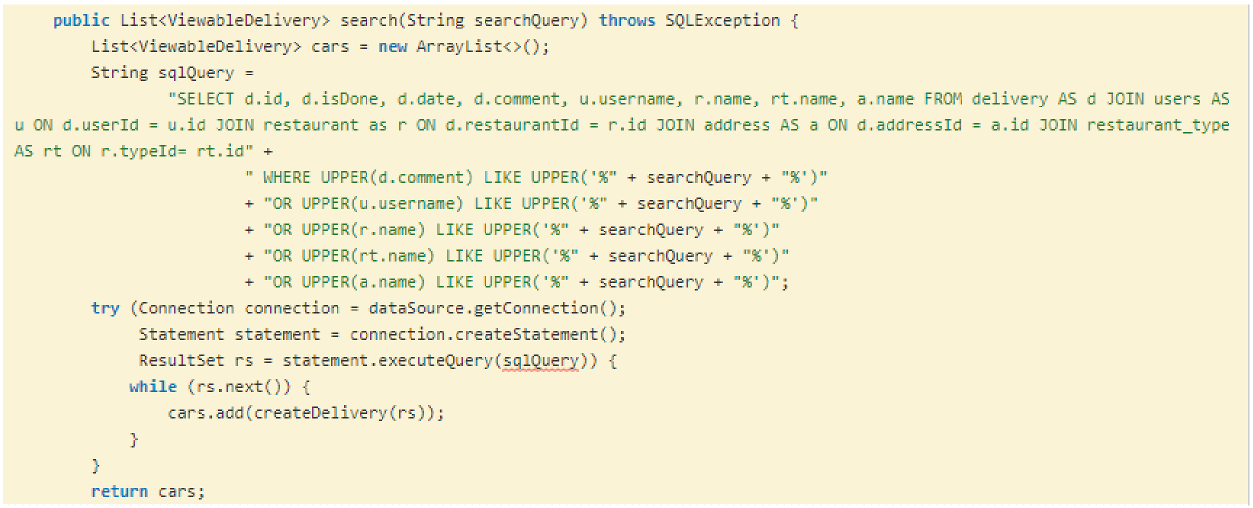
Kako postoji mogućnost da se izvrši SQL injection napad, kao na na primer klikom na link details, potrebno je da se upit zaštiti i da se koristi bezbedniji Statement, odnsno PreparedStatement.



```java
public List<DeliveryDetail> getDeliveryDetails(String id) {
    List<DeliveryDetail> details = new ArrayList<>();
    String sqlQuery = "SELECT f.id, di.amount, f.name, f.price FROM delivery_item AS di JOIN food AS f ON di.foodId = f.id WHERE deliveryId = ?";
    try (Connection connection = dataSource.getConnection();
          PreparedStatement statement = connection.prepareStatement(sqlQuery);) {

        statement.setString( parameterIndex: 1,id);
        ResultSet rs = statement.executeQuery();

        while (rs.next()) {
            details.add(createDetail(rs));
        }

    } catch (SQLException e) {
        e.printStackTrace();
    }
```



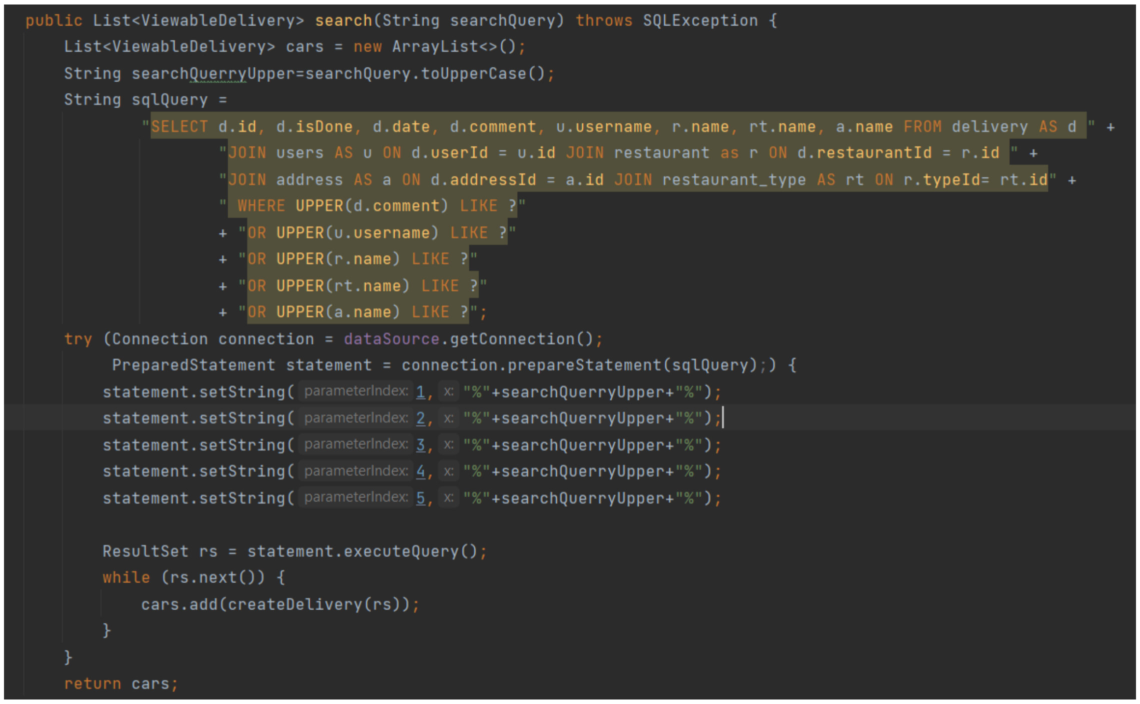
```java
public ViewableDelivery getDelivery(String id) {
    String sqlQuery = "SELECT d.id, d.isDone, d.date, d.comment, u.username, r.name, rt.name, a.name" +
            " FROM delivery AS d JOIN users AS u ON d.userId = u.id JOIN restaurant as r ON d.restaurantId = r.id " +
            "JOIN address AS a ON d.addressId = a.id JOIN restaurant_type AS rt ON r.typeId= rt.id WHERE d.id =?";

    try (Connection connection = dataSource.getConnection();
          PreparedStatement statement = connection.prepareStatement(sqlQuery);) {

        statement.setString( parameterIndex: 1,id);
         ResultSet rs = statement.executeQuery();

        if (rs.next()) {
            return createDelivery(rs);
        }

    } catch (SQLException e) {
        e.printStackTrace();
```



```
    public List<ViewableDelivery> search(String searchQuery) throws SQLException {
        List<ViewableDelivery> cars = new ArrayList<>();
        String sqlQuery =
                "SELECT d.id, d.isDone, d.date, d.comment, u.username, r.name, rt.name, a.name FROM delivery AS d JOIN users AS
u ON d.userId = u.id JOIN restaurant as r ON d.restaurantId = r.id JOIN address AS a ON d.addressId = a.id JOIN restaurant_type
AS rt ON r.typeId= rt.id" +
                        " WHERE UPPER(d.comment) LIKE UPPER('%" + searchQuery + "%')"
                        + "OR UPPER(u.username) LIKE UPPER('%" + searchQuery + "%')"
                        + "OR UPPER(r.name) LIKE UPPER('%" + searchQuery + "%')"
                        + "OR UPPER(rt.name) LIKE UPPER('%" + searchQuery + "%')"
                        + "OR UPPER(a.name) LIKE UPPER('%" + searchQuery + "%')";
        try (Connection connection = dataSource.getConnection();
             Statement statement = connection.createStatement();
             ResultSet rs = statement.executeQuery(sqlQuery)) {
            while (rs.next()) {
                cars.add(createDelivery(rs));
            }
        }
        return cars;
```

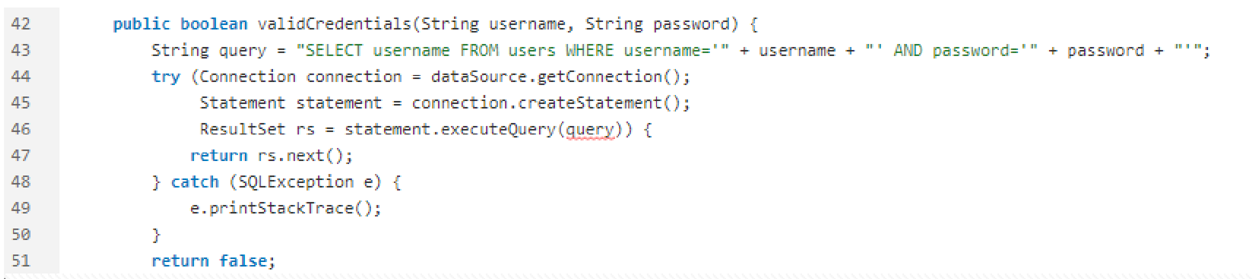
Status: true positive

Polje za pretragu je povezano direktno sa ovom funkcijom. Pošto korisnik ima mogućnost da unese slbodan tekst, upit mora biti zaštićen kako zlonamerni korisnik ne bi zloupotrebio pretragu.



```
public List<ViewableDelivery> search(String searchQuery) throws SQLException {
    List<ViewableDelivery> cars = new ArrayList<>();
    String searchQuerryUpper=searchQuery.toUpperCase();
    String sqlQuery =
            "SELECT d.id, d.isDone, d.date, d.comment, u.username, r.name, rt.name, a.name FROM delivery AS d " +
                    "JOIN users AS u ON d.userId = u.id JOIN restaurant as r ON d.restaurantId = r.id " +
                    "JOIN address AS a ON d.addressId = a.id JOIN restaurant_type AS rt ON r.typeId= rt.id" +
                    " WHERE UPPER(d.comment) LIKE ?"
                    + "OR UPPER(u.username) LIKE ?"
                    + "OR UPPER(r.name) LIKE ?"
                    + "OR UPPER(rt.name) LIKE ?"
                    + "OR UPPER(a.name) LIKE ?";
    try (Connection connection = dataSource.getConnection();
         PreparedStatement statement = connection.prepareStatement(sqlQuery);) {
        statement.setString( parameterIndex: 1, x: "%"+searchQuerryUpper+"%");
        statement.setString( parameterIndex: 2, x: "%"+searchQuerryUpper+"%");
        statement.setString( parameterIndex: 3, x: "%"+searchQuerryUpper+"%");
        statement.setString( parameterIndex: 4, x: "%"+searchQuerryUpper+"%");
        statement.setString( parameterIndex: 5, x: "%"+searchQuerryUpper+"%");

        ResultSet rs = statement.executeQuery();
        while (rs.next()) {
            cars.add(createDelivery(rs));
        }
    }
    return cars;
```

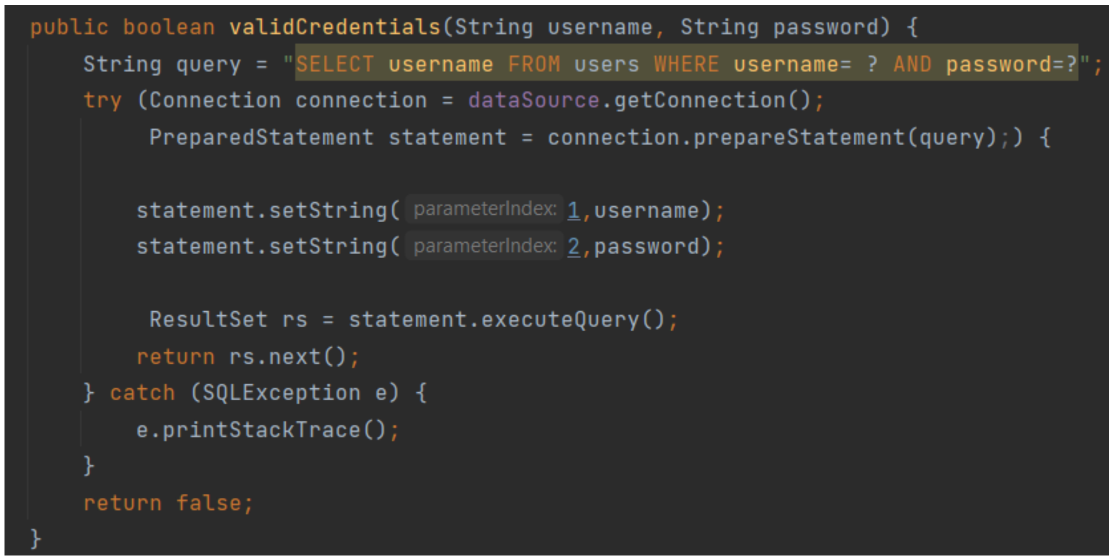
src/.../securesoftwaredevelopment/repository/UserRepository.java



```java
    public boolean validCredentials(String username, String password) {
        String query = "SELECT username FROM users WHERE username='" + username + "' AND password='" + password + "'";
        try (Connection connection = dataSource.getConnection();
             Statement statement = connection.createStatement();
             ResultSet rs = statement.executeQuery(query)) {
            return rs.next();
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return false;
```

Status: true positive

SQL injection napad je moguće uraditi sa forme za logovanje, te je upit u funkciji validCredentials potrebno zaštitit.

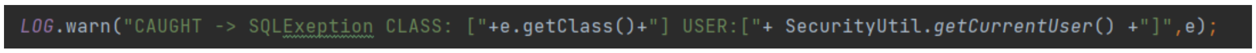


```java
public boolean validCredentials(String username, String password) {
    String query = "SELECT username FROM users WHERE username= ? AND password=?";
    try (Connection connection = dataSource.getConnection();
         PreparedStatement statement = connection.prepareStatement(query);) {

        statement.setString( parameterIndex: 1,username);
        statement.setString( parameterIndex: 2,password);

         ResultSet rs = statement.executeQuery();
        return rs.next();
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return false;
}
```

Preostale funkcije koje nemaju zaštićen upit (findUser,findByRoleId,findByUserId) su sigurne u smislu, ne pozivaju se od spolja, od stranje korisnika aplikacija, već od strane same apikacije tek nakon što su provereni kredencijali korisnika. Kako je upit validacije siguran, tako su i funkcije nakon validacije koje koriste unete kredencijale sigurne.

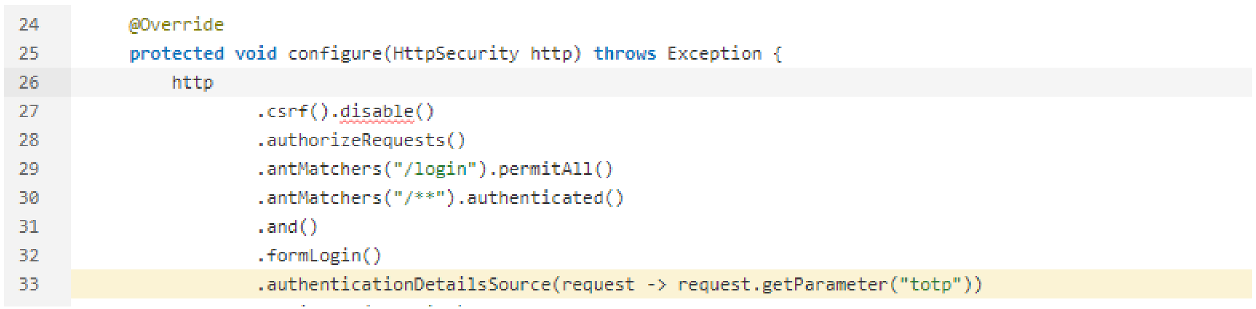
Security Hotspots: Insecure configuration

Security Hotspots sa nesigurnom konfiguracijom koji su detektovani od stane alata imaju isto upozorenje a ono je vezano za štampanje uhvaćenih izuzetaka, exception.printStackTrace() može otkriti neželjene informacije te se greška ispisuje u okviru loga greške.
Status upozorenja true positive.



```java
LOG.warn("CAUGHT -> SQLExeption CLASS: ["+e.getClass()+"] USER:["+ SecurityUtil.getCurrentUser() +"]",e);
```

CSRF



```
    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
                .csrf().disable()
                .authorizeRequests()
                .antMatchers("/login").permitAll()
                .antMatchers("/**").authenticated()
                .and()
                .formLogin()
                .authenticationDetailsSource(request -> request.getParameter("totp"))
```

Internet pretraživači imaju svoju, ugrađenu zaštitu od cors napada ali ona može biti zaobiđena, zato je potrebno implemetirati zaštitu od csrf napada koristeći csrf tokene ,stoga je **status upozorenja true positive.**