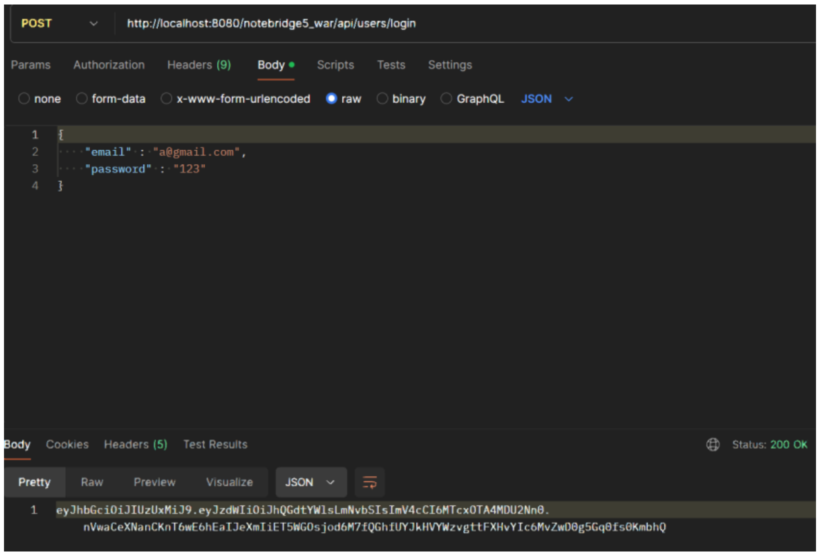
REPORT 3 Security

Notebridge5

JWT Token Security:

Except for signup and login, each endpoint requires a jwt token for authentication, which is provided by login endpoint if the email and password combination is matched in the database. For security reasons, passwords are stored as hashed versions. Additionally, we use salting to ensure a higher level of protection.

Here is an example login token generation by correct credentials:



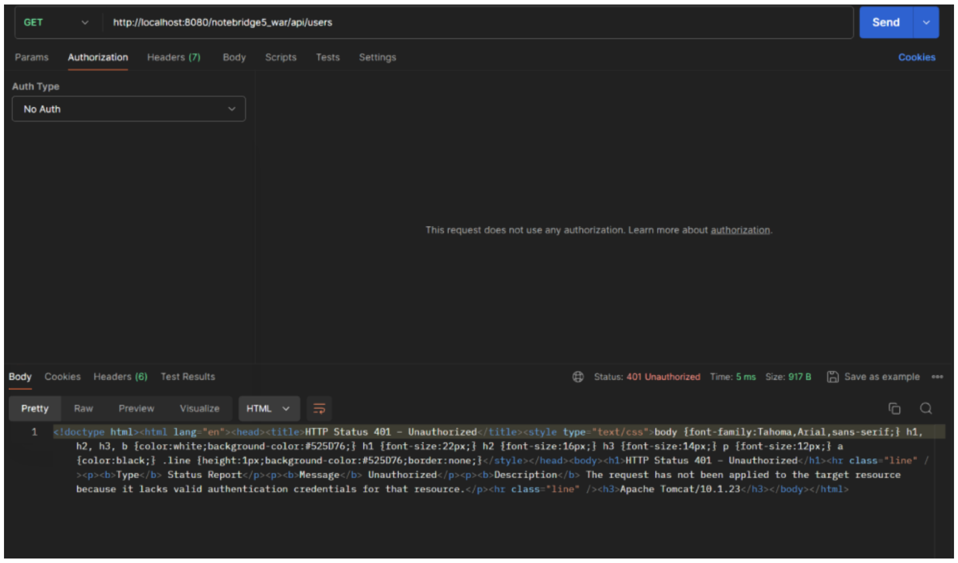
http://localhost:8080/notebridge5_war/api/users/login

```
{
    "email" : "a@gmail.com",
    "password" : "123"
}
```

Status: 200 OK

```
eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJhQGdtYWlsLmNvbSIsImV4cCI6MTcxOTA4MDU2Nn0.
    nVwaCeXNanCKnT6wE6hEaIJeXmIiET5WGOsjod6M7fQGhfUYJkHVYWzvgttFXHvYIc6MvZwD0g5Gq0fs0KmbhQ
```

Without this token, endpoints result in 401 unauthorized error:



http://localhost:8080/notebridge5_war/api/users

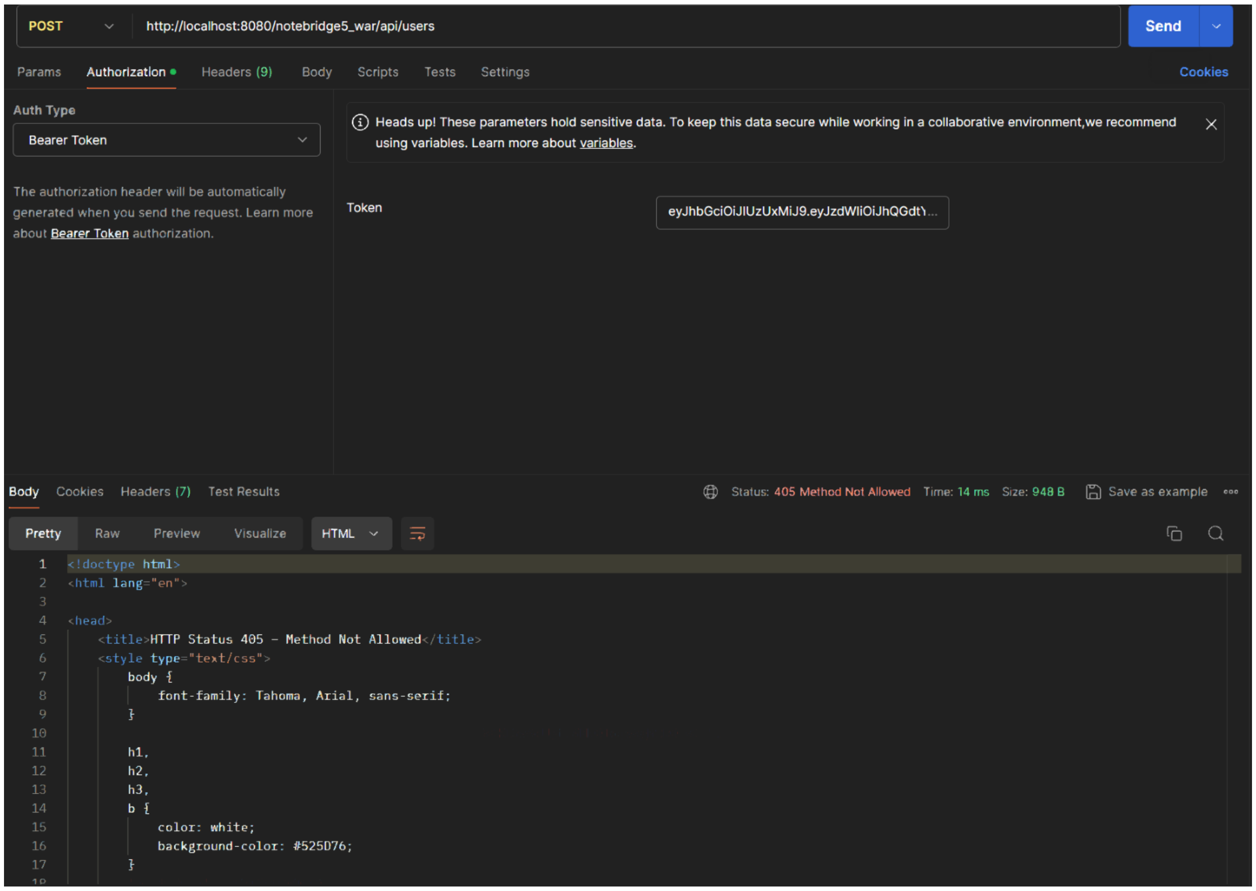
Auth Type: No Auth

This request does not use any authorization. Learn more about authorization.

Status: 401 Unauthorized Time: 5 ms Size: 917 B

```
<!doctype html><html lang="en"><head><title>HTTP Status 401 – Unauthorized</title><style type="text/css">body {font-family:Tahoma,Arial,sans-serif;} h1,
   h2, h3, b {color:white;background-color:#525D76;} h1 {font-size:22px;} h2 {font-size:16px;} h3 {font-size:14px;} p {font-size:12px;} a
   {color:black;} .line {height:1px;background-color:#525D76;border:none;}</style></head><body><h1>HTTP Status 401 – Unauthorized</h1><hr class="line" /
   ><p><b>Type</b> Status Report</p><p><b>Message</b> Unauthorized</p><p><b>Description</b> The request has not been applied to the target resource
   because it lacks valid authentication credentials for that resource.</p><hr class="line" /><h3>Apache Tomcat/10.1.23</h3></body></html>
```

If the endpoint and method mismatch, 405 method not allowed error is given:



http://localhost:8080/notebridge5_war/api/users

Auth Type: Bearer Token

The authorization header will be automatically generated when you send the request. Learn more about Bearer Token authorization.

Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment,we recommend using variables. Learn more about variables.

Token: eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJhQGdt\...

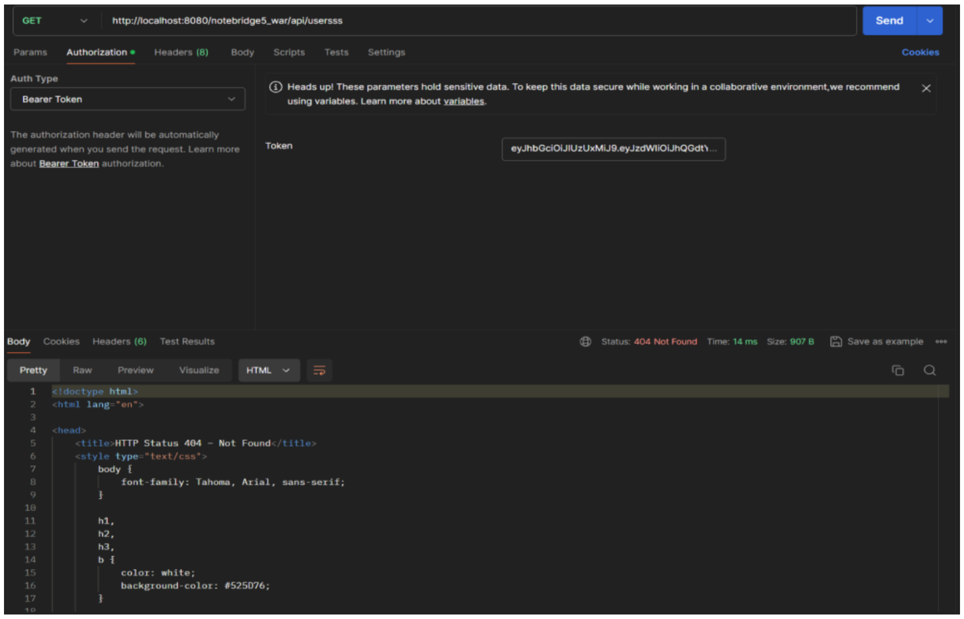
Status: 405 Method Not Allowed Time: 14 ms Size: 948 B

```
<!doctype html>
<html lang="en">

<head>
    <title>HTTP Status 405 – Method Not Allowed</title>
    <style type="text/css">
        body {
            font-family: Tahoma, Arial, sans-serif;
        }

        h1,
        h2,
        h3,
        b {
            color: white;
            background-color: #525D76;
        }
```

If the endpoint has typo or if the endpoint does not exist, it results in 404 not found error:


GET http://localhost:8080/notebridge5_war/api/usersss

Auth Type: Bearer Token

The authorization header will be automatically generated when you send the request. Learn more about Bearer Token authorization.

Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment,we recommend using variables. Learn more about variables.

Token: eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJhQGdt\...

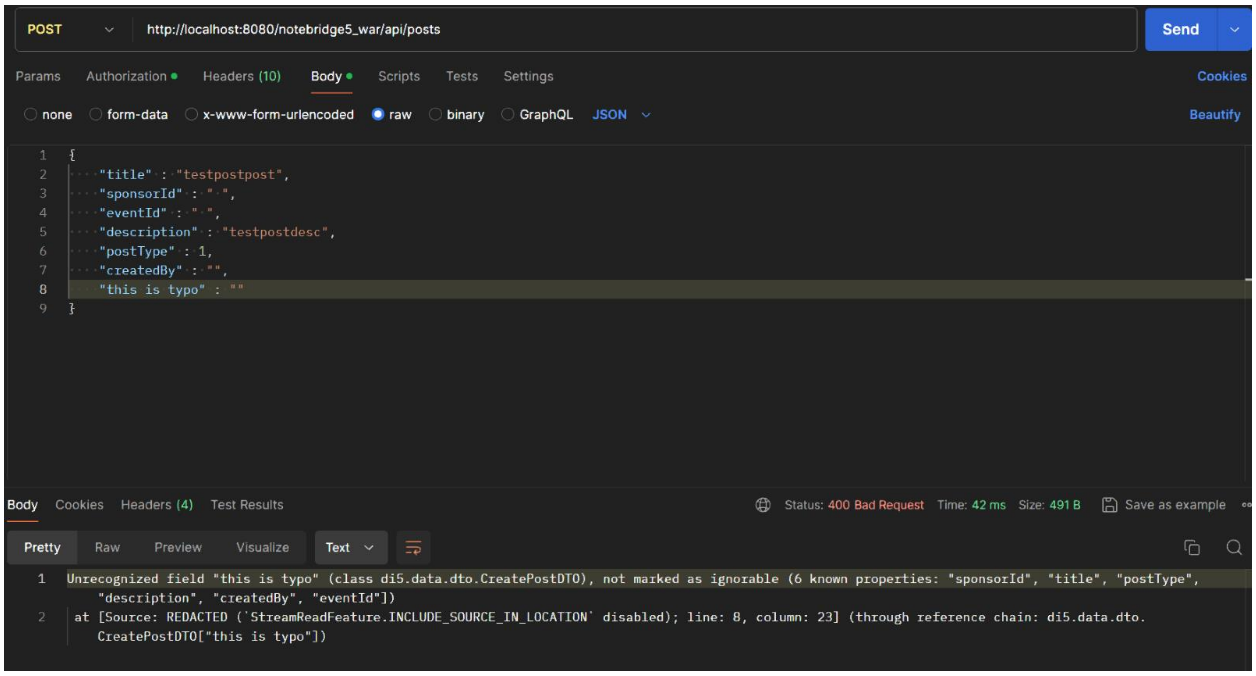
Status: 404 Not Found Time: 14 ms Size: 907 B

```
<!doctype html>
<html lang="en">

<head>
    <title>HTTP Status 404 – Not Found</title>
    <style type="text/css">
        body {
            font-family: Tahoma, Arial, sans-serif;
        }

        h1,
        h2,
        h3,
        b {
            color: white;
            background-color: #525D76;
        }
```

If the request body is not in appropriate format, 400 bad request error is given:

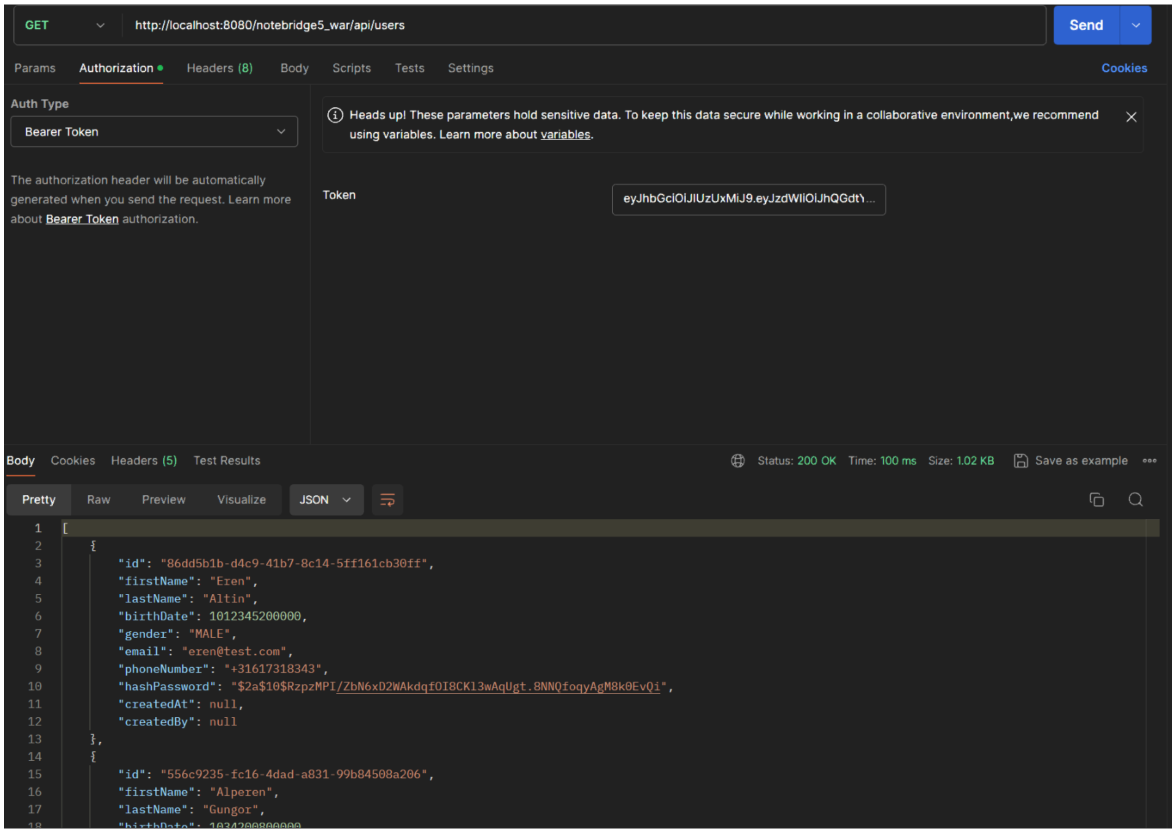

POST http://localhost:8080/notebridge5_war/api/posts

```
{
    "title" : "testpostpost",
    "sponsorId" : " ",
    "eventId" : " ",
    "description" : "testpostdesc",
    "postType" : 1,
    "createdBy" : "",
    "this is typo" : ""
}
```

Status: 400 Bad Request Time: 42 ms Size: 491 B

```
Unrecognized field "this is typo" (class di5.data.dto.CreatePostDTO), not marked as ignorable (6 known properties: "sponsorId", "title", "postType",
    "description", "createdBy", "eventId"])
 at [Source: REDACTED (`StreamReadFeature.INCLUDE_SOURCE_IN_LOCATION` disabled); line: 8, column: 23] (through reference chain: di5.data.dto.
    CreatePostDTO["this is typo"])
```

If the endpoint and the method is written correctly, 200 ok results in expected values:


GET http://localhost:8080/notebridge5_war/api/users

Auth Type: Bearer Token

The authorization header will be automatically generated when you send the request. Learn more about Bearer Token authorization.

Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment,we recommend using variables. Learn more about variables.

Token: eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJhQGdt...

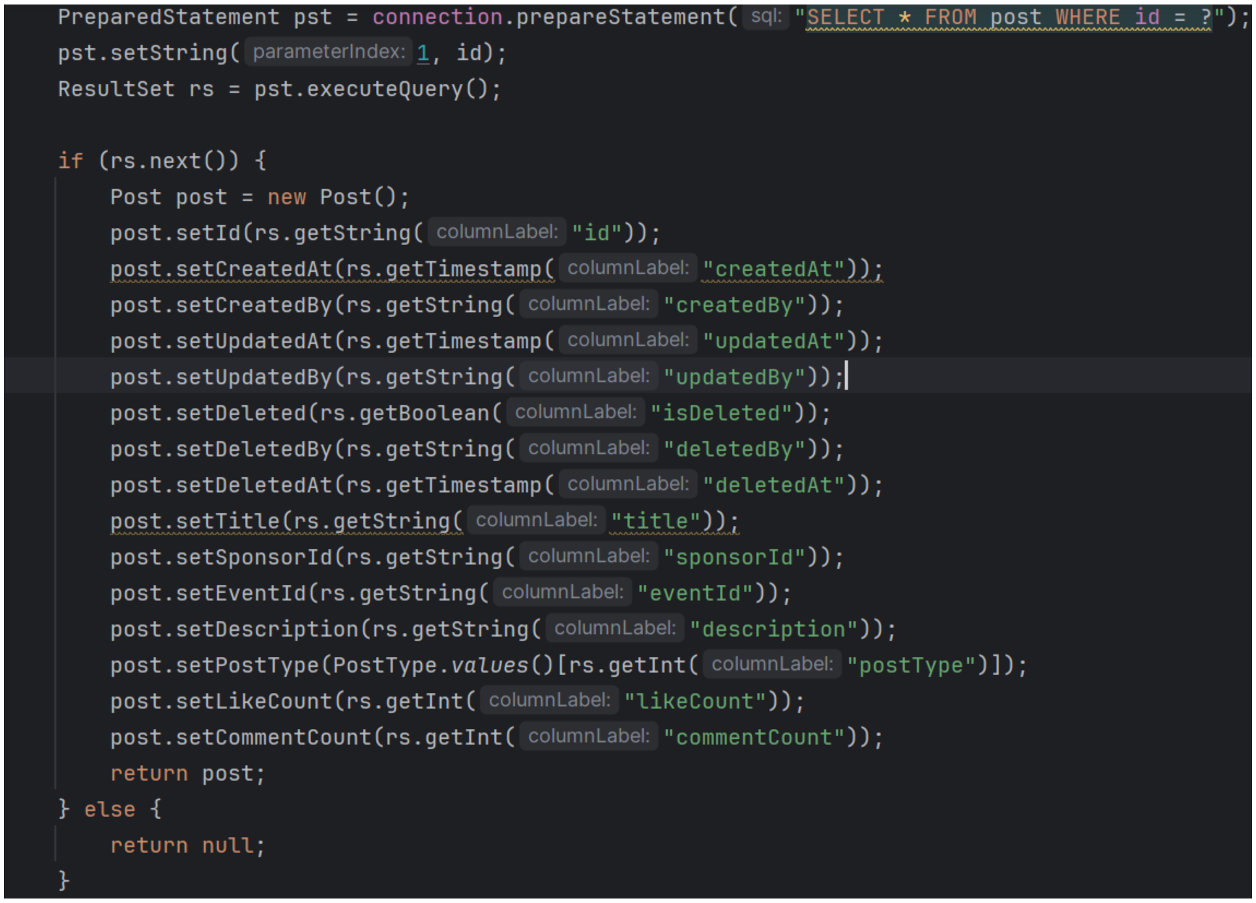
Status: 200 OK Time: 100 ms Size: 1.02 KB

```
[
    {
        "id": "86dd5b1b-d4c9-41b7-8c14-5ff161cb30ff",
        "firstName": "Eren",
        "lastName": "Altin",
        "birthDate": 1012345200000,
        "gender": "MALE",
        "email": "eren@test.com",
        "phoneNumber": "+31617318343",
        "hashPassword": "$2a$10$RzpzMPI/ZbN6xD2WAkdqfOI8CKl3wAqUgt.8NNQfoqyAgM8k0EvQi",
        "createdAt": null,
        "createdBy": null
    },
    {
        "id": "556c9235-fc16-4dad-a831-99b84508a206",
        "firstName": "Alperen",
        "lastName": "Gungor",
        "birthDate": 1034200800000
```

PREPARED STATEMENTS:

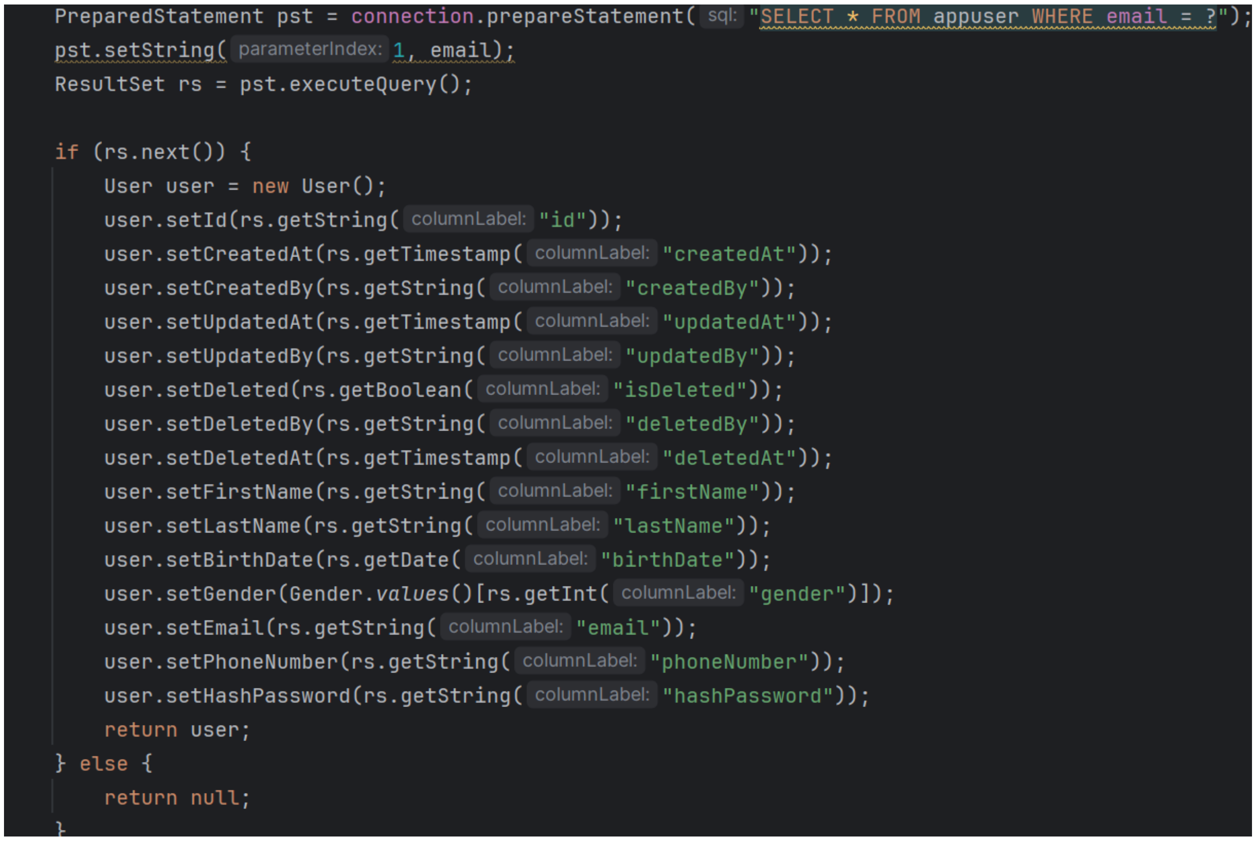
In order to prevent attacks and provide a higher level of security, every time we communicated with the database we used prepared statements. This helped us to protect against SQL injection attacks by ensuring that SQL code is properly separated from data inputs. This method allows us to safely handle user inputs by using parameterized queries, where user-supplied data is treated as parameters rather than executable code. By implementing these measures, we ensure the integrity of our data and protect our application from potential vulnerabilities.

Some example code is given below:



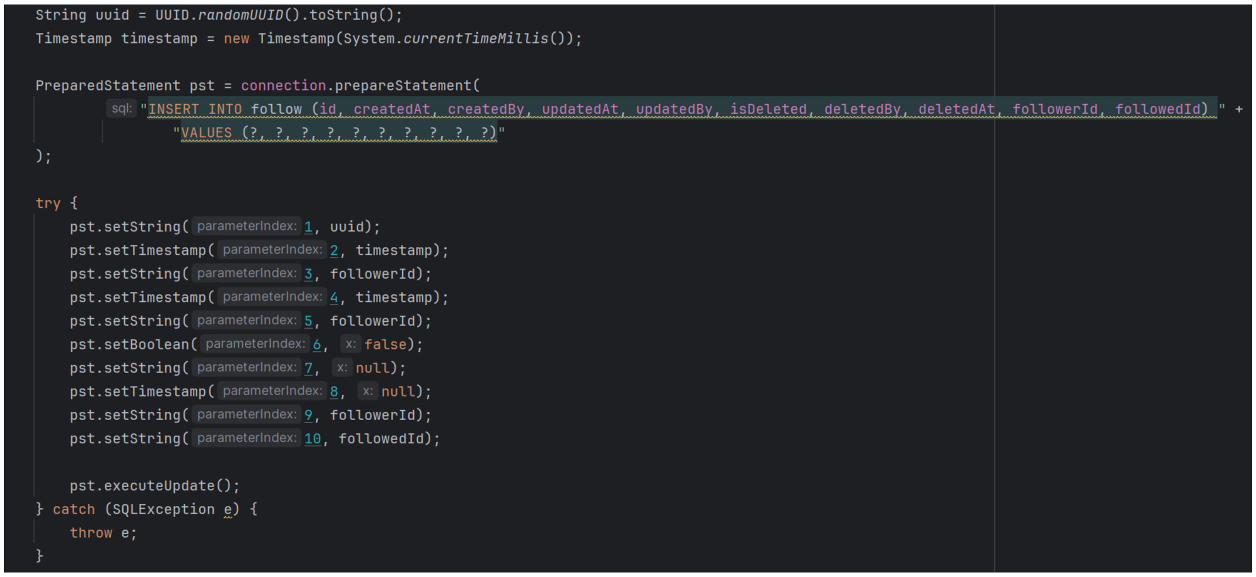
```java
    PreparedStatement pst = connection.prepareStatement( sql: "SELECT * FROM post WHERE id = ?");
    pst.setString( parameterIndex: 1, id);
    ResultSet rs = pst.executeQuery();

    if (rs.next()) {
        Post post = new Post();
        post.setId(rs.getString( columnLabel: "id"));
        post.setCreatedAt(rs.getTimestamp( columnLabel: "createdAt"));
        post.setCreatedBy(rs.getString( columnLabel: "createdBy"));
        post.setUpdatedAt(rs.getTimestamp( columnLabel: "updatedAt"));
        post.setUpdatedBy(rs.getString( columnLabel: "updatedBy"));
        post.setDeleted(rs.getBoolean( columnLabel: "isDeleted"));
        post.setDeletedBy(rs.getString( columnLabel: "deletedBy"));
        post.setDeletedAt(rs.getTimestamp( columnLabel: "deletedAt"));
        post.setTitle(rs.getString( columnLabel: "title"));
        post.setSponsorId(rs.getString( columnLabel: "sponsorId"));
        post.setEventId(rs.getString( columnLabel: "eventId"));
        post.setDescription(rs.getString( columnLabel: "description"));
        post.setPostType(PostType.values()[rs.getInt( columnLabel: "postType")]);
        post.setLikeCount(rs.getInt( columnLabel: "likeCount"));
        post.setCommentCount(rs.getInt( columnLabel: "commentCount"));
        return post;
    } else {
        return null;
    }
```



```
    PreparedStatement pst = connection.prepareStatement( sql: "SELECT * FROM appuser WHERE email = ?");
    pst.setString( parameterIndex: 1, email);
    ResultSet rs = pst.executeQuery();

    if (rs.next()) {
        User user = new User();
        user.setId(rs.getString( columnLabel: "id"));
        user.setCreatedAt(rs.getTimestamp( columnLabel: "createdAt"));
        user.setCreatedBy(rs.getString( columnLabel: "createdBy"));
        user.setUpdatedAt(rs.getTimestamp( columnLabel: "updatedAt"));
        user.setUpdatedBy(rs.getString( columnLabel: "updatedBy"));
        user.setDeleted(rs.getBoolean( columnLabel: "isDeleted"));
        user.setDeletedBy(rs.getString( columnLabel: "deletedBy"));
        user.setDeletedAt(rs.getTimestamp( columnLabel: "deletedAt"));
        user.setFirstName(rs.getString( columnLabel: "firstName"));
        user.setLastName(rs.getString( columnLabel: "lastName"));
        user.setBirthDate(rs.getDate( columnLabel: "birthDate"));
        user.setGender(Gender.values()[rs.getInt( columnLabel: "gender")]);
        user.setEmail(rs.getString( columnLabel: "email"));
        user.setPhoneNumber(rs.getString( columnLabel: "phoneNumber"));
        user.setHashPassword(rs.getString( columnLabel: "hashPassword"));
        return user;
    } else {
        return null;
    }
```



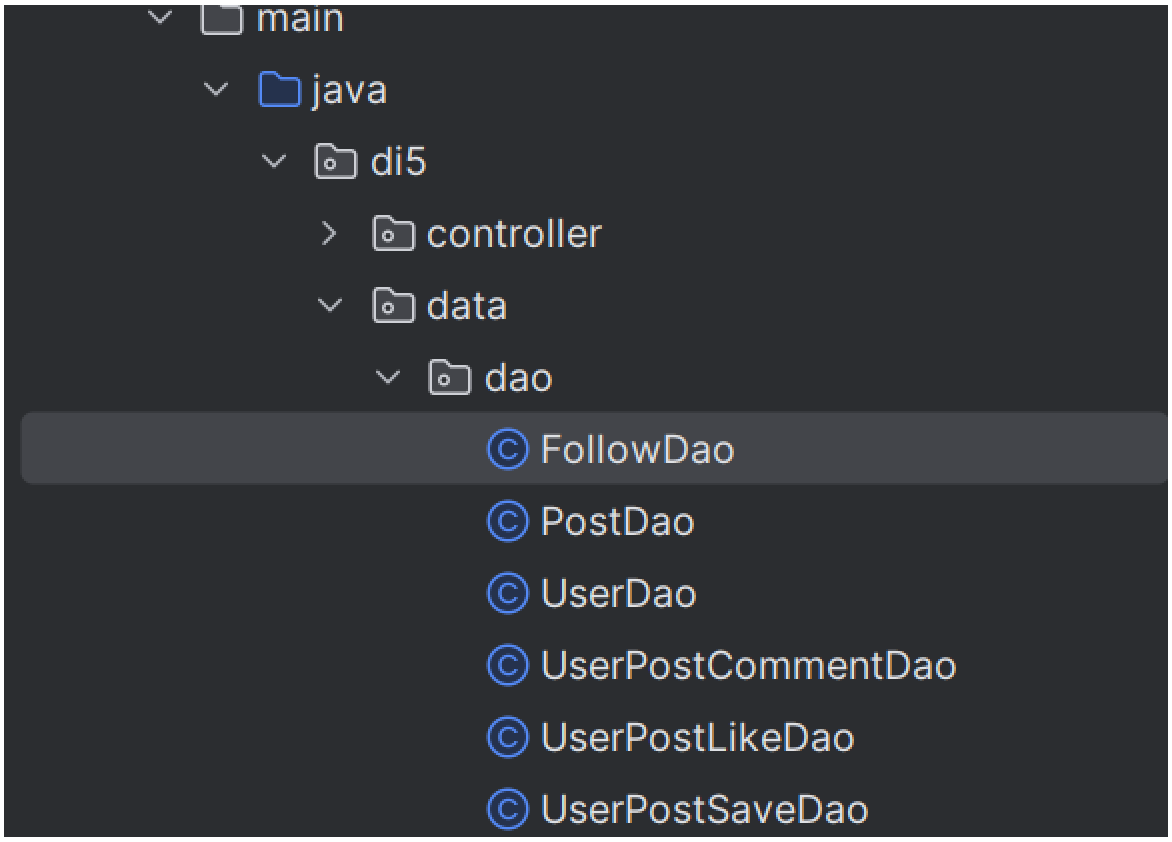
```
String uuid = UUID.randomUUID().toString();
Timestamp timestamp = new Timestamp(System.currentTimeMillis());

PreparedStatement pst = connection.prepareStatement(
        sql: "INSERT INTO follow (id, createdAt, createdBy, updatedAt, updatedBy, isDeleted, deletedBy, deletedAt, followerId, followedId) " +
                "VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)"
);

try {
    pst.setString( parameterIndex: 1, uuid);
    pst.setTimestamp( parameterIndex: 2, timestamp);
    pst.setString( parameterIndex: 3, followerId);
    pst.setTimestamp( parameterIndex: 4, timestamp);
    pst.setString( parameterIndex: 5, followerId);
    pst.setBoolean( parameterIndex: 6, x: false);
    pst.setString( parameterIndex: 7, x: null);
    pst.setTimestamp( parameterIndex: 8, x: null);
    pst.setString( parameterIndex: 9, followerId);
    pst.setString( parameterIndex: 10, followedId);

    pst.executeUpdate();
} catch (SQLException e) {
    throw e;
}
```

Communication with database is done using data access objects (DAO's). Codes above are part of DAO's, where the full list that we implemented so far is below:



Every query that we have used in these DAO's are parameterized.

Additionally, necessary constraints on tables are also enforced on the database.

All the code are tested (details can be found in the Assignment 3 Method Report) for different situations to make sure there are no functionality or security issues with the app.

In addition to prepared statements, we enforced some basic rules on database to sanitize input even further in necessary parts.