

Obligatory 3: Web Security

1 Introduction

The goal of this assignment is to evaluate the security of the given 'vulnerable' web applications and implement mitigations or controls for eliminating or reducing some of the identified security risks. The web applications by default, are vulnerable to a number of web security risks.

There are two web applications for this assignment: DAT152BlogApp and DAT152WebSearch. **DAT152WebSearch** has been implemented as an Identity Provider (IdP) with specific OAuth endpoints and OpenID authentication flow.

DAT152BlogApp has been implemented as a Service Provider (SP) where a client can also have a single sign-on (SSO) experience.

You are provided with the source code for the two applications located as a zipped file on canvas, and you will be required to download, import, and run the applications in your own environment. The **DAT152WebSearch** is using the Derby DBMS which is a DB engine written entirely in Java with a disk-space footprint of ~3.5MB. The database **SecOblig** and tables - **AppUser** and **History** are created by default during container initialization. Be aware that the data do not persist after you might have stopped the web server or when the server synchronises your code changes. The application has been tested using Apache Tomcat v9.0.22 and Java version "12.0.2" 2019-07-16 and "15.0.2" 2021-01-19.

You will need to import the web applications into your IDE environment as **Maven** projects and ensure that they are running.

DAT152WebSearch URL - <http://localhost:9092/DAT152WebSearch>

DAT152BlogApp URL - <http://localhost:9091/blogapp>

1.1 Description of DAT152WebSearch web application

The web application provides a simple search functionality against an "exclusive" English glossary (localhost or external). In order to use the application, a user must be registered and be logged in. A default admin user (**admin**) with password (**password**) is created during initialization.

1.1.1 Functionalities of the application for normal user privilege

- To use the application, a user must be registered.
- A user can perform search of words in the dictionary after she/he must have logged in.
- A user can see the list of searches she/he has performed (i.e., history of own searches) and sort by date or search word.
- A user is able to see the dictionary server (default selection) upon registration (stored as a cookie).

1.1.2 Additional functionalities for admin level privilege

In addition to the above features:

- An admin can see the top five (5) previous searches done by the application independent of any user.
- An admin can elevate the privilege of any user to admin level

1.1.3 The application contains the following security functionalities

- User information must be viewed as sensitive data and should not be viewable by others.
- It should not be possible to use the dictionary without being logged in as a registered user.
- In addition, only a logged in user should be able to change/update his password from My Details page.
- Only the admin has the access right to elevate the privilege of another user.

1.1.4 Identity Provider functionalities

- There are 4 endpoints provided: Register, Authorize, Token, and UserInfo.
- **Register:** A client must first call the Register endpoint to register its app. The IdP returns a unique "client_id" as a json response.
- **Authorize:** Clients are redirected to this endpoint for authorization flow.
- **Token:** When a client is authenticated and completed the authorization flow, both the id_token, access_token, and refresh_token are issued from this endpoint. In addition, an access_token can be re-issued based on client's request that is accompanied with the refresh_token. In this assignment, we are only using the id_token.
- **UserInfo:** Clients can send a request to this endpoint to obtain user information from the IdP. Client's request must contain the access_code in the request authorization header.

2 Description of DAT152BlogApp web application

The web application provides a single page where a user can post comments. In order to use the application, a user must be registered with the application and then log in. Alternatively, a user can use SSO by logging via DAT152WebSearch IdP.

The purpose of this web app is to use it for testing JWT authentication token and access token weaknesses in SSO environment.

2.1 Functionalities of the application for normal user privilege

- To use the application, a user must be registered locally or be registered with the IdP (DAT152WebSearch) and obtain a client_id.
- When a user login via SSO, three tokens: id_token, access_token, and refresh_token are generated. You can find them in the backend console.
- A user can enter comments after she/he must have logged in.
- A user can see all the comments of other users

2.2 Additional functionalities for admin level privilege

In addition to the above features:

- An SSO admin can delete all the comments on the blog.

3 Using IdP endpoints by clients

3.1 Register endpoint

Request: curl -X POST http://localhost:9092/DAT152WebSearch/register --data phone=120389734562'

Response: json object containing unique clientId and phone

```
{
  "clientId":"7759FCCB4EC2445EF13E1516F9CDB650",
  "phone":"120389734562"
}
```

4 The task

The task is divided into two parts. **Part 1** – Identifying vulnerabilities and **Part 2** – Mitigating Vulnerabilities.

Vulnerabilities that **MUST** be identified:

OWASP Web Security Testing Guide (WSTG) v4.2

	Test Category	Vulnerability	Target application
1	4.4.7 WSTG-ATHN-07: Testing for Weak Password Policy (pg.159)	Password/Session management	DAT152WebSearch
2	4.7.5 WSTG-INPV-05: Testing for SQL Injection (pg.234)	SQL injection (SQLi)	DAT152WebSearch
3	4.7.1 WSTG-INPV-01: Testing for Reflected Cross Site Scripting (pg.217)	Cross-site scripting (XSS) - Reflected	DAT152WebSearch
4	4.7.2 WSTG-INPV-02: Testing for Stored Cross Site Scripting (pg.223)	Cross-site scripting (XSS) - Stored	DAT152WebSearch
5	4.6.5 WSTG-SESS-05: Testing for Cross Site Request Forgery (pg.200)	Cross site request forgery (CSRF)	DAT152WebSearch
6	SSO OpenID authentication token (JWT)	Elevation of privilege and weak token validation	DAT152WebBlogApp

Tools/APIs: You may use some of the tools/APIs we have discussed in class such as Spotbug/FindSecBugs, OWASP ZAP, and ESAPI.

Part 1. – Identifying vulnerabilities

In this task, you are REQUIRED to identify at least ONE security defect each under the listed vulnerabilities in the web applications (**DAT152WebSearch** and **DAT152WebBlogApp**). You will describe in addition to the vulnerability, how you have exploited the found vulnerabilities. Finally, you will provide examples of possible consequences of the vulnerabilities based on your proof.

You will document your results in this part in a structured report (see Report section).

Part 2. – Mitigating vulnerabilities

You are required to fix all the six vulnerabilities. This implies that you MUST write control code to mitigate these vulnerabilities in the part of the code where you found them. In the case of

CSRF, you SHOULD implement either the “**Synchronizer Token Pattern**” or the “**Double submit cookie pattern**”.

You will document your results in this part in a structured report (see Report section)

5 Report

You are required to write a detailed and structured report for each part of the task. In addition, you MUST include every necessary result from the tools that you have used as an appendix in your report document.

5.1 Part 1 - Identifying vulnerabilities

Use the template below to document your results. The report MUST follow the format specified below.

Additionally, you can provide pictures and screenshots of your results (e.g., application at runtime) in the applicable sections.

Vulnerability #1: WSTG-ATHN-07: Testing for Weak Password Policy

Description:

Possible consequences:

File (s): ExampleFile2.java

Code: code where vulnerability exists

Payload:

Technique: How did you find the weakness? Manual, DAST, or SAST

Vulnerability #2: WSTG-INPV-05: Testing for SQL Injection

Description:

Possible consequences:

File (s):

Code:

Payload:

Analysis Technique:

Vulnerability #3: WSTG-INPV-01: Testing for Reflected Cross Site Scripting

Description:

Possible consequences:

File (s):

Code:

Payload:

Analysis Technique:

Vulnerability #4: WSTG-INPV-02: Testing for Stored Cross Site Scripting

Description:

Possible consequences:

File (s):

Code:

Payload:

Analysis Technique:

Vulnerability #5: WSTG-SESS-05: Testing for Cross Site Request Forgery

Description:

Possible consequences:

File (s):

Code:

Payload:

Analysis Technique:

Vulnerability #6: SSO OpenID authentication token (JWT)

Description: Weaknesses in JWT authentication token - **DAT152WebBlogApp**

Possible consequences:

Guiding questions/Answer format:

- What is the id_token (authentication token) used to authenticate to the DAT152WebBlogApp?
 - **Answer:** Paste your own token in the answer
- Where is the id_token (authentication token) stored in the client environment?
 - **Answer:**
- What are the vulnerabilities that you think exist in this id_token both from the IdP and SP endpoints?
 - **Answer:** (Mention the weaknesses you think exist in the creation of the id_token at the IdP and the verification at the SP)
- What security decisions are being made using this id_token?
 - **Answer:** (Mention what this id_token is used for within authentication and authorization)
- Can a user elevate his privilege in this id_token?
 - **Answer:** Show specific proof of your modifications and how you have exploited it.

Possible consequences:

File (s):

Code:

Payload:

Analysis Technique:

5.2 Part 2 - Mitigating vulnerabilities

Use the template below to document the mitigation/control you have provided for all the vulnerabilities. Describe what the control is doing and show testcases for validating the control. The report MUST follow the below format. Additionally, you can provide screenshots of your results (e.g., application at runtime).

Vulnerability #1: WSTG-ATHN-07: Testing for Weak Password Policy

Description:

Part of code (fixes):

Mitigation/control code:

Vulnerability #2: WSTG-INPV-05: Testing for SQL Injection

Description:

Part of code:

Mitigation/control code:

Vulnerability #3: WSTG-INPV-01: Testing for Reflected Cross Site Scripting

Description:

Part of code:

Mitigation/control code:

Vulnerability #4: WSTG-INPV-02: Testing for Stored Cross Site Scripting

Description:

Part of code:

Mitigation/control code:

Vulnerability #5: WSTG-SESS-05: Testing for Cross Site Request Forgery

Description:

Part of code:

Mitigation/control code:

Vulnerability #6: SSO OpenID authentication token (JWT)

Description:

Part of code:

Description: SSO and weakness in JWT authentication token

Mitigation/control code:

- What vulnerabilities exist in this id_token from the IdP and SP endpoints?
 - **Mitigation:** Must fix in the “authorizationCodeRequest” method in the “Token.java” class in DAT152WebSearch (IdP) and the “RequestHelper.java” in the DAT152WebBlogApp (SP) and paste your code fixes here

6 Submission

You are required to submit the following on Canvas and within the deadline

- a single and complete report in **pdf** format.
- a zipped file of your modified code for both web applications which must be runnable.

Please be aware that INCOMPLETE work will not be approved. All the six (6) vulnerabilities MUST be identified and fixed to get approval. Submission that fails to follow the reporting format will be outrightly rejected. Finally, it must be possible to verify your code. Code that does not run due to unfixed bugs will be returned and the work rejected.

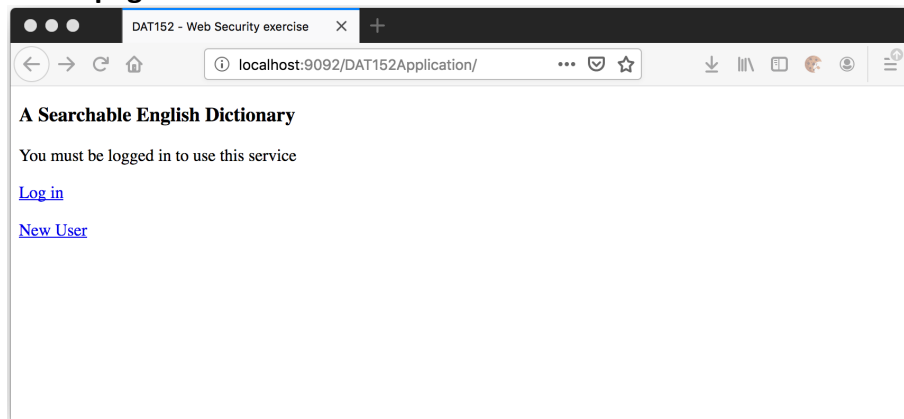
7 Appendix I – DAT152WebSearch Application Forms

URL: <http://localhost:9092/DAT152WebSearch>

This appendix shows the eight (8) forms contained in the web application.

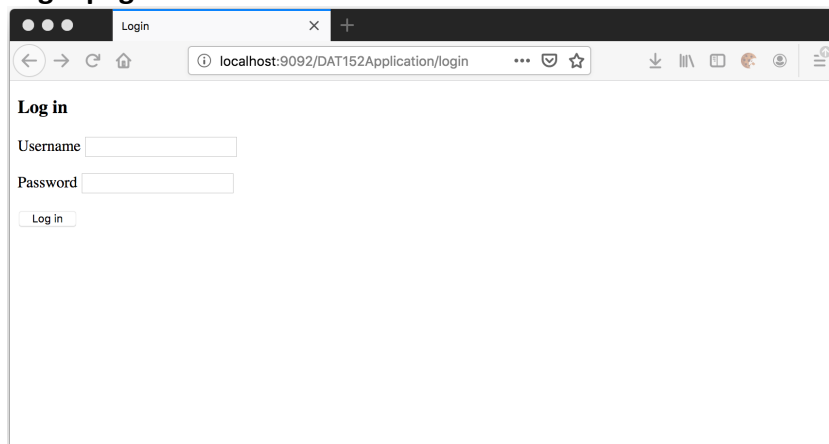
Home page, Login page, New user page, Update user password page, Main search page, Search result page, Personal detail page, and Update role page.

Home page



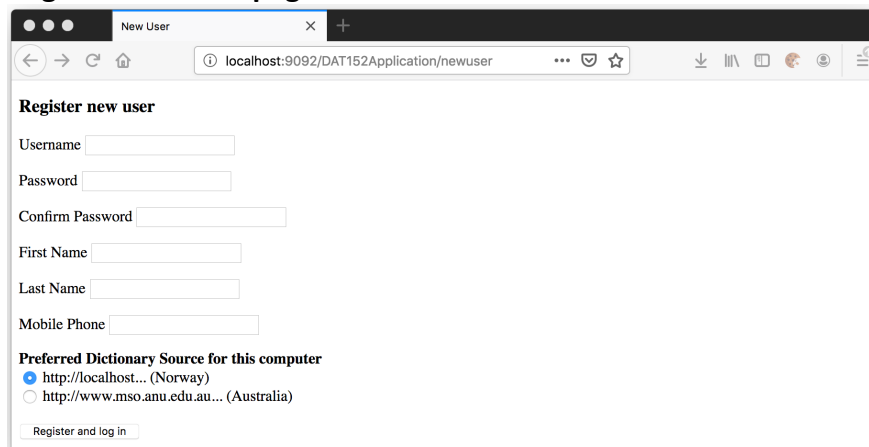
The screenshot shows a web browser window with the title "DAT152 - Web Security exercise". The address bar displays "localhost:9092/DAT152Application/". The main content area has the heading "A Searchable English Dictionary" and a message "You must be logged in to use this service". Below the message are two links: "Log in" and "New User".

Login page



The screenshot shows a web browser window with the title "Login". The address bar displays "localhost:9092/DAT152Application/login". The main content area has the heading "Log in" and two input fields: "Username" and "Password". Below the input fields is a "Log in" button.

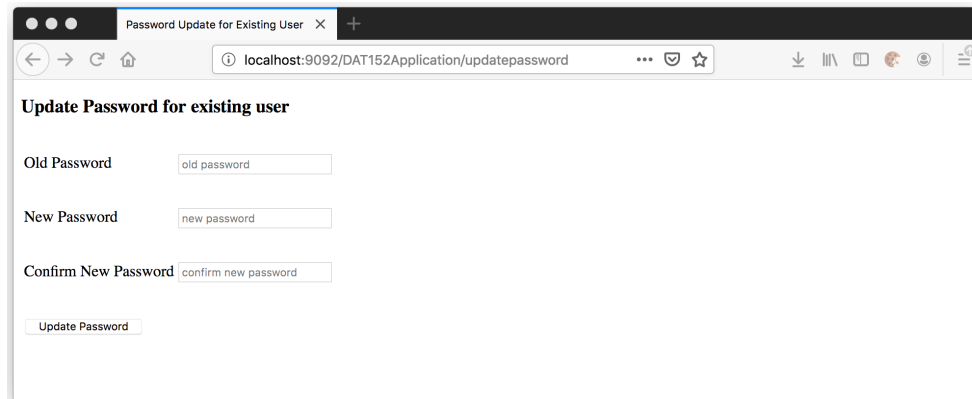
Register new user page



The screenshot shows a web browser window with the title "New User". The address bar displays "localhost:9092/DAT152Application/newuser". The main content area has the heading "Register new user" and several input fields: "Username", "Password", "Confirm Password", "First Name", "Last Name", and "Mobile Phone". Below the input fields is a section titled "Preferred Dictionary Source for this computer" with two radio buttons: "http://localhost... (Norway)" (selected) and "http://www.mso.anu.edu.au... (Australia)". At the bottom is a "Register and log in" button.

Update user password page

Here, a current user can update his/her password.



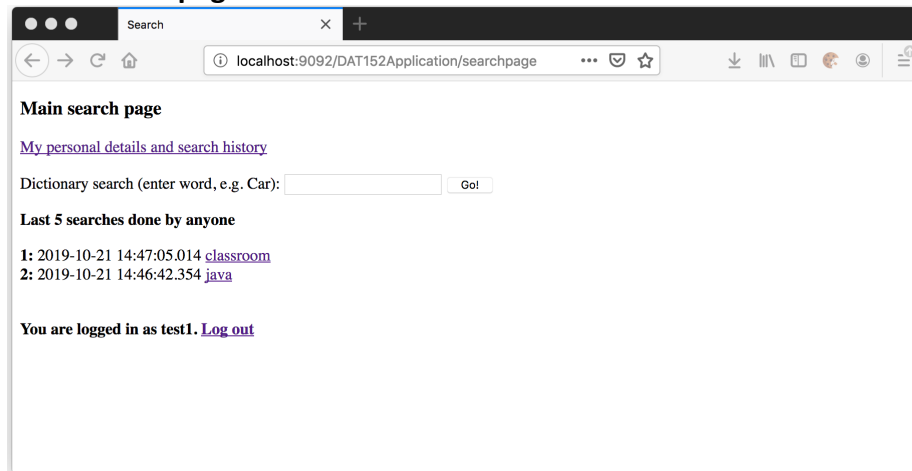
Update Password for existing user

Old Password

New Password

Confirm New Password

Main search page



Main search page

[My personal details and search history](#)

Dictionary search (enter word, e.g. Car):

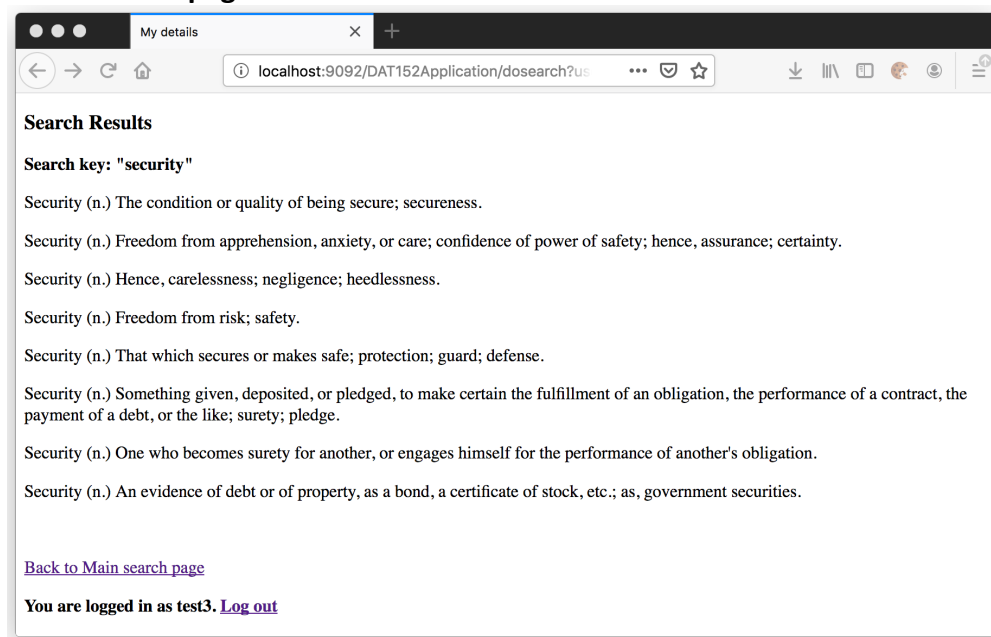
Last 5 searches done by anyone

1: 2019-10-21 14:47:05.014 [classroom](#)

2: 2019-10-21 14:46:42.354 [java](#)

You are logged in as test1. [Log out](#)

Search result page



Search Results

Search key: "security"

Security (n.) The condition or quality of being secure; secureness.

Security (n.) Freedom from apprehension, anxiety, or care; confidence of power of safety; hence, assurance; certainty.

Security (n.) Hence, carelessness; negligence; heedlessness.

Security (n.) Freedom from risk; safety.

Security (n.) That which secures or makes safe; protection; guard; defense.

Security (n.) Something given, deposited, or pledged, to make certain the fulfillment of an obligation, the performance of a contract, the payment of a debt, or the like; surety; pledge.

Security (n.) One who becomes surety for another, or engages himself for the performance of another's obligation.

Security (n.) An evidence of debt or of property, as a bond, a certificate of stock, etc.; as, government securities.

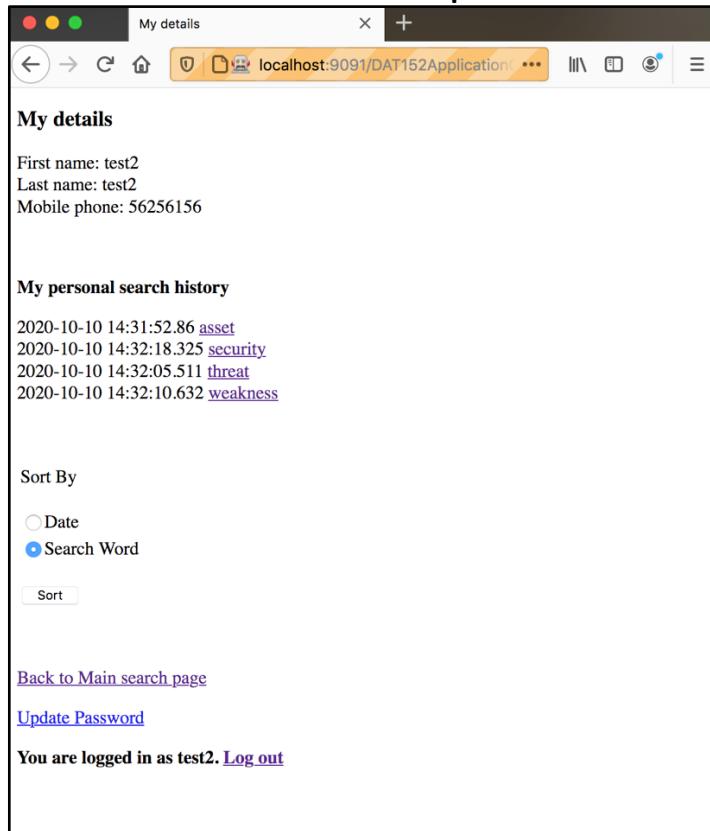
[Back to Main search page](#)

You are logged in as test3. [Log out](#)

Personal detail page (Normal user)

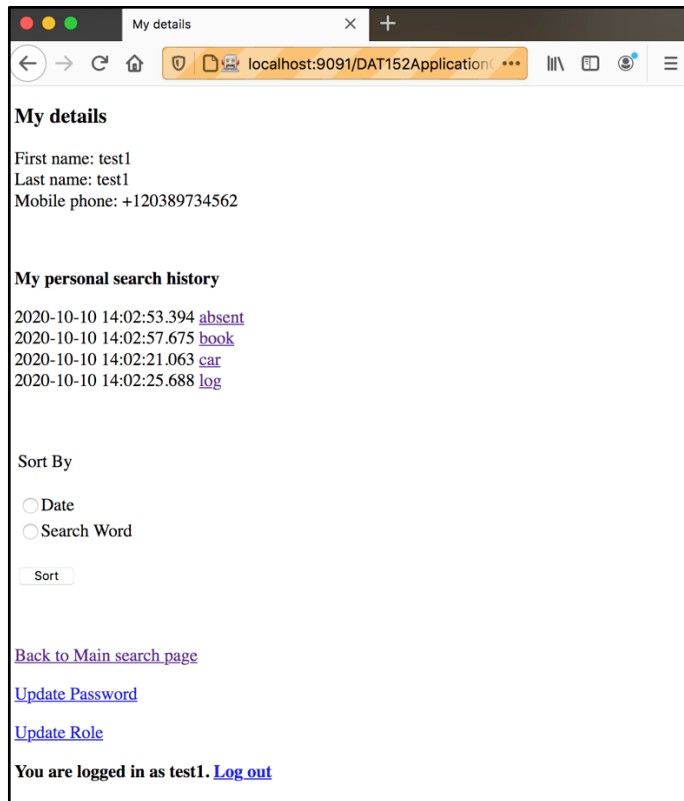
This page shows the personal details including all personal search history for a normal user.

Note: Normal user can't see the **Update Role** function



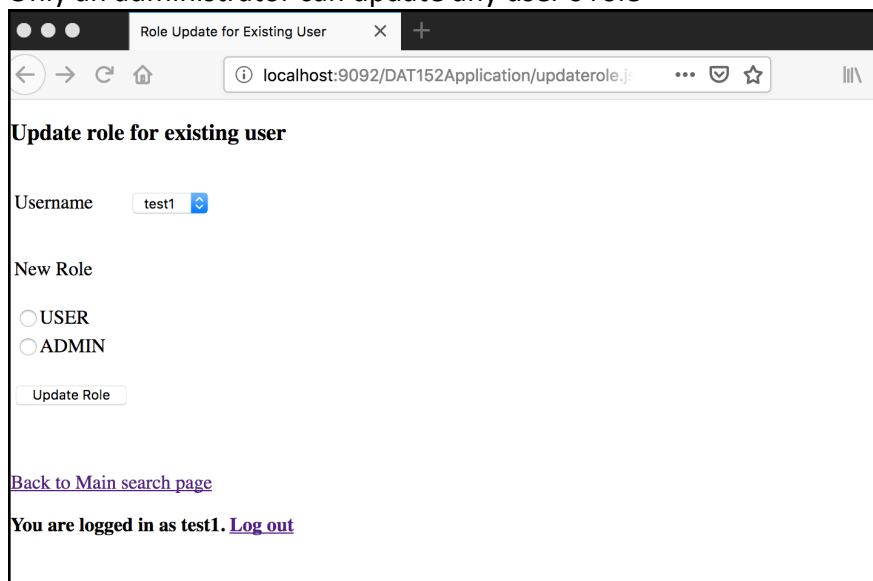
Personal detail page (admin)

An admin can see the Update Role function in addition to other features



Update Role page

Only an administrator can update any user's role




8 Appendix II – DAT152BlogWebApp Application Forms

URL: <http://localhost:9091/blogapp>

Login page

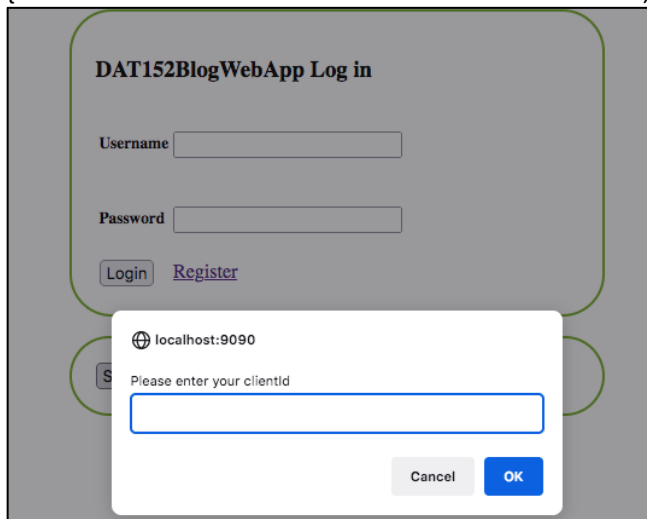


Register page (Local user)

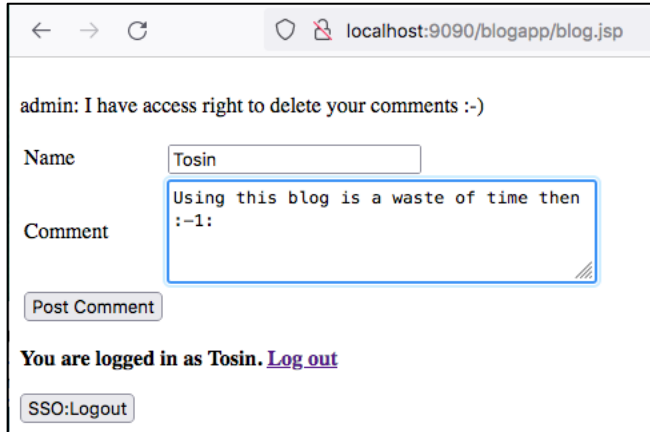


SSO Login (You must obtain clientId from the IdP register endpoint as follows:)

```
$ curl -X POST http://localhost:9092/DAT152WebSearch/register --data phone='120389734562' {"clientId":"9BBC13CF5998CF6E7FFAB8A62D69ABF9","phone":"120389734562"}
```



Normal user view



A screenshot of a web browser window showing the 'Normal user view' of a blog application. The browser's address bar displays 'localhost:9090/blogapp/blog.jsp'. The page content includes a message from the admin: 'admin: I have access right to delete your comments :-)', a form for posting a comment with the name 'Tosin' and the comment text 'Using this blog is a waste of time then :-1:', a 'Post Comment' button, a login status message 'You are logged in as Tosin. [Log out](#)', and an 'SSO:Logout' button.

← → ↻ localhost:9090/blogapp/blog.jsp

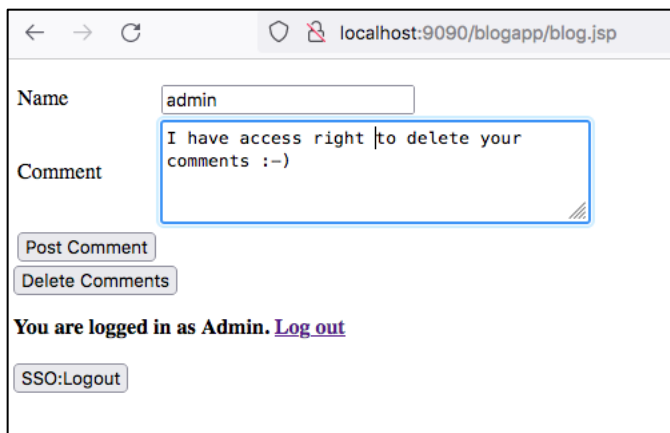
admin: I have access right to delete your comments :-)

Name

Comment

You are logged in as Tosin. [Log out](#)

Admin view



A screenshot of a web browser window showing the 'Admin view' of the same blog application. The browser's address bar displays 'localhost:9090/blogapp/blog.jsp'. The page content includes a form for posting a comment with the name 'admin' and the comment text 'I have access right to delete your comments :-)', 'Post Comment' and 'Delete Comments' buttons, a login status message 'You are logged in as Admin. [Log out](#)', and an 'SSO:Logout' button.

← → ↻ localhost:9090/blogapp/blog.jsp

Name

Comment

You are logged in as Admin. [Log out](#)