

# Performance Evaluation of UNSWBook

Callum Bannister

Jerome Samir

Peter Bishay

Aaron Quan

Waseem Sajeev

October 22, 2017

# Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Application Details</b>	<b>3</b>
2.1	User Login . . . . .	3
2.2	Search . . . . .	4
2.3	Adding a friend . . . . .	4
2.4	Posting a message . . . . .	5
<b>3</b>	<b>Test Setup</b>	<b>6</b>
3.1	Server . . . . .	6
3.2	Client . . . . .	6
3.3	Experimental Method . . . . .	7
<b>4</b>	<b>Results</b>	<b>8</b>
<b>5</b>	<b>Conclusion</b>	<b>9</b>

# 1. Introduction

Maintaining quality of service under heavy usage is essential for any web application that expects a large amount of concurrent users. As there is currently no demand for an asocial media platform, we have performed a series of experiments on a selection of functions of our UNSWBook application to obtain data on performance degradation as the number of simultaneous users increases. We present the methodology; what functions were tested and what hardware were they tested on. Then, we present results of the experiment, an analysis of the results, and the conclusions to be drawn.

## 2. Application Details

We looked at four basic functionalities of UNSWBook for performance under pressure. They are as follows:

1. User login
2. Searching for friends
3. Adding a friend
4. Posting a message

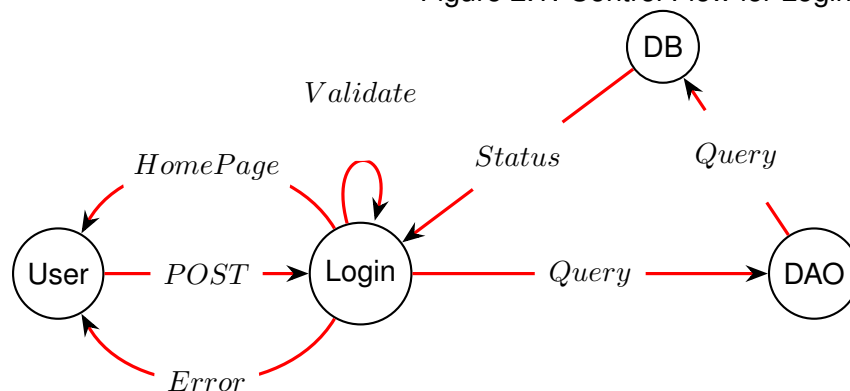
For each of these functions, we will explain the implementation and what database interactions were required, in order to give context to the performance data in Chapter 3.

### 2.1 User Login

The user submits their username and password with a POST request. The request body is passed to the *Login* servlet's *doPost* method, where we perform initial validation of the input, checking that the required fields exist.

Then, we construct the DAO object for a User. We make a single query to the database to check that a non-admin user with the name and password exists, and we retrieve their unique id and whether or not they are banned. Then, we route the user to either the login page, or their home page, returning an error message where necessary.

Figure 2.1: Control Flow for Login

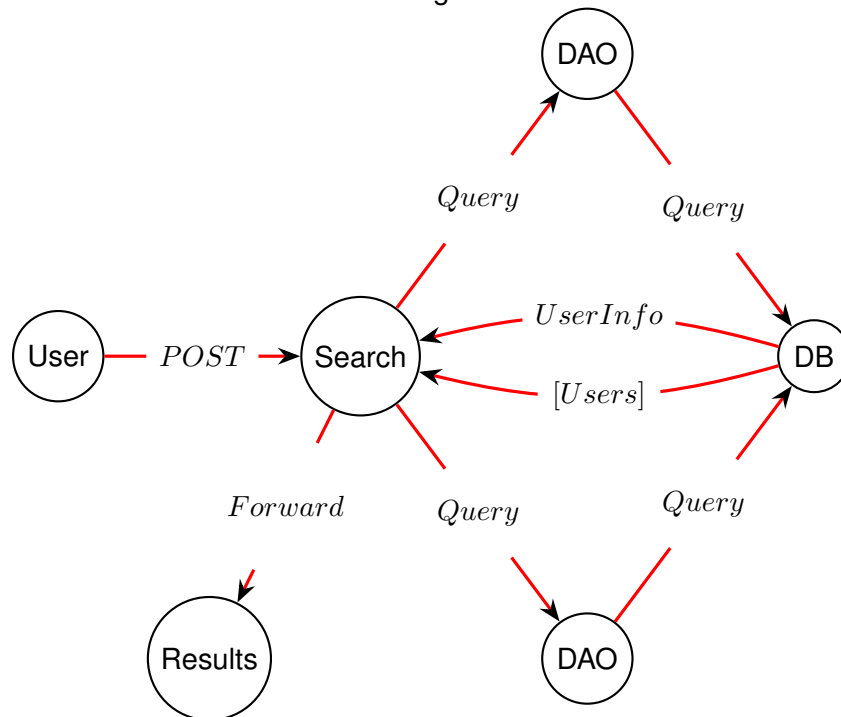


## 2.2 Search

In this section we describe only the basic search functionality. We expect that the advanced search to be used relatively infrequently, so optimisation of that function would be of secondary priority.

As before, the user submits the search query information via POST, getting routed to the appropriate method in the *Search* servlet. First, we construct a DAO object for the current logged in user, querying the database to obtain their information. Then, we construct a second DAO object, and use it to retrieve a list of users whose username contains the submitted string. We set this as an attribute, and forward the request onto *results.jsp*.

Figure 2.2: Control Flow for Search



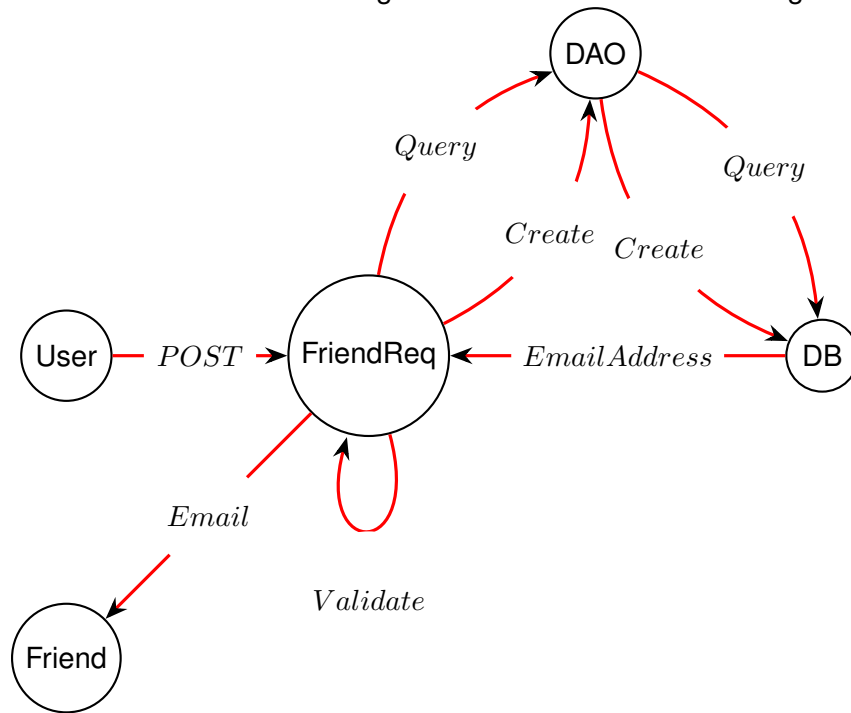
## 2.3 Adding a friend

Although adding a friend consists of two operations, the need for a confirmation email to link the two makes measuring them as a conjoined unit unrealistic. Instead, we present only the sending of a friend request.

The friend request submits the user id for the requester and the requestee, dispatched to the *FriendReq* servlet. As was the case for *Login*, we perform simple validation of the parameters. Then, we look up the requestee by their id, constructing the DAO and performing the query to the database through it.

We add a table to the *friends* database, with the confirmed flag set to *false*, requiring a second query. Afterwards, we extract their email, and send a message containing the appropriate link to confirm the friend request.

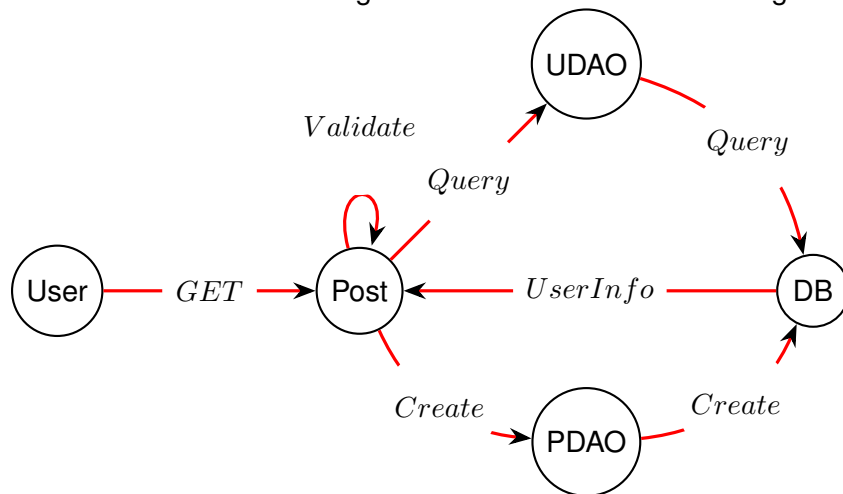
Figure 2.3: Control Flow for Adding a Friend



## 2.4 Posting a message

Unlike the other operations, the user posts a message by sending a GET request. This is routed to the *Post* servlet, where we once again perform simple validation on the input. We construct the DAO object and look up the currently logged in user as before. We then create a *post* DAO object, and use it to create the appropriate table in the database.

Figure 2.4: Control Flow for Posting a Message



## 3. Test Setup

In this chapter we present the hardware, setup, and experimental procedure that we used to obtain the data.

### 3.1 Server

The application server had the following hardware specifications:

Table 3.1: Server Hardware

Hardware	Specification
HDD	A Floppy Disk I Found In An Abandoned Building With The Word "Spooky" On It
RAM	12MB
CPU	Literally A Potato

The machine was running Windows 3.1. We used Tomcat 7.0.3 as our server, and Apache Derby 10.9.1 as our database. To maximise the accuracy of our results the only user processes on the machine were the Tomcat and Derby servers.

The connection pool size was kept at the default for the jdbc driver (a *maximum* of 100 and *minimum* of 10). We did not predict that manipulating this value would lead to significantly different results.

### 3.2 Client

The client had the following hardware specifications:

Table 3.2: Client Hardware

Hardware	Specification
HDD	A Floppy Disk I Found In An Abandoned Building With The Word "Spooky" On It
RAM	12MB
CPU	Literally A Potato

As with the server machine, extraneous processes were kept to a minimum. Both the client and server were on the same LAN, minimising factors unrelated to the performance of UNSWBook from affecting the results.

### 3.3 Experimental Method

We constructed our experiment to simulate the following scenario:

1. A user logs in, searches for a new friend, adds them, and posts a message to acknowledge their new friendship

We added a delay of five seconds between each user action. We tested this scenario for increasing numbers of concurrent users: 5,10,20,25,40,50,75,80,100 and 200. Before each test attempt, we cleared the database of any existing posts and/or friends, in order to keep those factors constant.

However, the experiment cooked the potato, which was subsequently eaten by a passing hungry dog.



## **4. Results**

### **4.1 Data**

### **4.2 Analysis**

As we had no results, we instead lit a candle inside a goat's skull, and asked Moloch, the Stone Owl to give us some data. He said 'good job'.

## **5. Conclusion**

We recommend not consorting with dark powers beyond the grave for experimental data, as the walls now bleed, and a soft humming noise that causes neighbourhood cats to fall out of trees can be heard.