Table of Contents

1. Website Overview

The user is greeted by the homepage where he can choose to either register or login to an account. If the user chooses to register an account, he is expected to input the required fields correctly and the outcome is indicated on the registration message page after the information has been submitted.

If the user already possesses an account, he is required to input the correct credentials in the login form which is found on the homepage. After three consecutive invalid logins, the user account is locked for five minutes. After a successful login, the user is redirected to the betting page, where he can place bets until reaching the account limit or until the user decides to log out.

2. Levels of Testing

Different levels of testing are performed on the system:

- Unit testing – unit tests cover most of the logic of the website
- Integration testing – Integration tests test the database connection and database queries. These tests are performed on a test database (set through a properties file)
- User acceptance testing – User stories are tested using cucumber and selenium

3. Testing Methodologies Used

Dependency Injection was employed to inject dependent components into test objects. Both setter injection and constructor injection were used, depending on the case. The use of this methodology required additional setter or constructor methods.

4. Testing strategies

The testing strategies used are the random strategy and the partitioning strategy. Such examples where these strategies are used are given below.

- Random Strategy – This strategy was used to test credit card numbers, since there are no particular boundaries.
- Partitioning Strategy – This strategy was used to test the function that validates a login request, when an account is locked. This has only two partitions: the difference in time from the moment the account was locked is greater than five minutes or smaller than five minutes.

5. Coverage Report

The following image shows the code coverage statistics as measured by the Emma plugin for Eclipse. The overall code coverage is 82.3%.

Assignment	90.6 %	7,678	799	8,477
src/main/java	82.3 %	2,146	460	2,606
com.assignment.DBObjects	64.7 %	626	341	967
User.java	65.0 %	487	262	749
Bet.java	63.8 %	139	79	218
com.assignment.functionalities	89.8 %	316	36	352
BettingImp.java	80.7 %	151	36	187
LoginImp.java	100.0 %	78	0	78
RegistrationImpl.java	100.0 %	87	0	87
com.assignment.servlets	91.6 %	284	26	310
RegisterServlet.java	90.1 %	109	12	121
BettingServlet.java	94.5 %	121	7	128
LoginServlet.java	88.5 %	54	7	61
com.assignment.util	72.7 %	64	24	88
MenuImpl.java	56.1 %	23	18	41
Props.java	71.4 %	15	6	21
MessagePageImpl.java	100.0 %	26	0	26
com.assignment.validations	96.0 %	529	22	551
RegistrationValidationImp.java	95.0 %	416	22	438
BettingValidationsImp.java	100.0 %	67	0	67
LoginValidationsImpl.java	100.0 %	46	0	46
com.assignment.mongodb	94.6 %	192	11	203
MongoDBConnectionWrapperImpl.ja	75.0 %	21	7	28
MongoDBWrapperImpl.java	96.8 %	122	4	126
MongoDBActionsWrapperImpl.java	100.0 %	49	0	49
com.assignment.requests	100.0 %	135	0	135
BettingRequestsImpl.java	100.0 %	63	0	63
UserRequestImpl.java	100.0 %	72	0	72
src/test/java	94.2 %	5,532	339	5,871

Illustration 1: Code coverage metrics

The table on the next page explains why certain code was not covered.

User	65.0 %
hashCode()	0.0 %
equals(Object)	59.9 %
setCreditcard(String)	0.0 %
setCvv(String)	0.0 %
setDob(String)	0.0 %
setExpdate(String)	0.0 %
setSname(String)	0.0 %

Illustration 2: Coverage stats for class User

Package Name	Class Name	Method Name	Reason
DBObjects	User Bet	hashCode	These are the default methods generated by the IDE, hence assumed correct. Also, these methods are rarely used
		equals	
		Various setter methods	These are not covered as they are simple setter methods where no calculations are performed
Functionalities	BettingImp	addBet	This method could not be tested because the return value depends on an instance of type BettingRequestImpl which is created within. Also, this method does not have any decision statements or calculations to perform, it relies on methods of the aforementioned instance.
		getMessage	Simply a getter
Servlets	RegisterServlet BettingServlet LoginServlet	init	This method is used on initialization of the servlet, and its sole purpose is to generate instances of the variables used. To test these classes, mocks are set through setter methods
Util	MenuImpl	getLoggedInMenu GetLoggedOutMenu	The purpose of these methods is to return an HTML string which represent a menu
Validations	RegistrationValidationsImp	Various methods	The only lines of code that are not tested perform null checking
Mongodb	MongoDbConnectionWrapperImpl	connect	Exception handling is not tested as the instantiation of a new object cannot be mocked

6. Web Testing using Selenium Test Suite

The Page Object approach was adopted to increase the readability of the code. Each web page has a corresponding class, composed of methods which are responsible for manipulating the respective page object. Each of these classes are initialized using a WebDriver instance to be able to perform operations in that browser. In order to be able to access HTML fields using selenium, every HTML field was given a unique id. Elements are accessed by their id since this is the most efficient method.

7. CUCUMBER ALTERATIONS

The following cucumber scenarios where altered:

- Scenario 2: Incorrect registration data
    - Since on our site the error messages are placed within an HTML `<div>` which has a unique id, this scenario had to be slightly altered to make it possible for the story to provide the id of the HTML element in which the error message appears. Hence, the field “<errorMessage>” was added. Also, the actual error message needed to be added so that it could be asserted against the actual message that is shown in the browser. The updated scenario is as follows:

```
Scenario Outline: Change field names

Given I am a user trying to register
When I fill in a form with correct data and I change the "<fieldname>" field to have incorrect input
Then I should be told in "<errorMessage>" that the data in "<fieldname>" is "<incorrect>"
Examples:
|fieldname    |incorrect                  |errorMessage    |
|firstName    |Invalid characters         |name_error      |
|lastName     |Invalid characters         |surname_error   |
|dob          |Please enter date of birth |dob_error       |
|creditcard   |Invalid card               |creditcard_error|
|expiry_date  |Invalid Expirary date      |expiry_error    |
```

- Scenario 7: Verify that free users can only place low-risk bets
    - When a user makes a bet, a message with the outcome of the bet is displayed. For this reason, this message is stated in the user story so that an assertion against this message could be performed. The modified scenario:

```
Scenario Outline: invalid risks

Given I am a user with a free account
When I try to place a "<risk>" bet of 5 euros
Then I should see "<message>"
Examples:
|risk   |message                  |
|low    |Bet placed successfully  |
|medium |Invalid risk             |
|high   |Invalid risk             |
```

8. Model Based Testing

8.1 Model Design

The model is composed of states and actions. Every page in the website is represented by a state in the model. Transitions from one page to another and the operations that happen during such transitions, such as submitting information, are represented by the actions. During execution of the model, the current state is obtained from the current URL of the browser. For example, if the website is currently on the page http://localhost:8080/Assignment/betting.jsp, the current state is identified as the Betting page.

The model we designed is composed of the following states:

Home page	Registration	Registration Message Page
Login	Login Error Page	Betting Page

Below is the description of each action:

Action	Description
Register	Navigate to the registration page
Submit Details	Fill the registration form and submit it. The user has a 75% probability of being a free user and 25% of being a premium user.
Proceed to Login	Navigate to the login page
Valid Login	A correct username and password are provided with a probability of 75%, and the user is directed to the Betting Page.
Invalid Login	Wrong username and password are inputted with a probability of 25% and the user is redirected to the Login Error Page.
Place Bet	Place bet with a probability of 50%. If the user is free, a random bet between 0 and 6 is placed, while if the user is a premium one, he will place a random bet between 100 and 2000. While the user does not choose to log out, he will remain on the betting page.
Logout	Navigate to the Home Page with a probability of 50%

8.2 Model Overview

The model simulates the typical behavior of a user who is using the website for the first time. The model is initiated by loading the homepage. The user must first register to be able to log in. When registration is complete, the details are submitted and the user is redirected to the Registration message page, which notifies the user with the outcome of the registration. The user then proceeds to the login page. Given the case that incorrect credentials are inputted, the user is redirected to the Login Error Page, from where he must return to the login page. When the user provides correct credentials, he will be redirected to the Betting page, where he can either place a bet or logout.

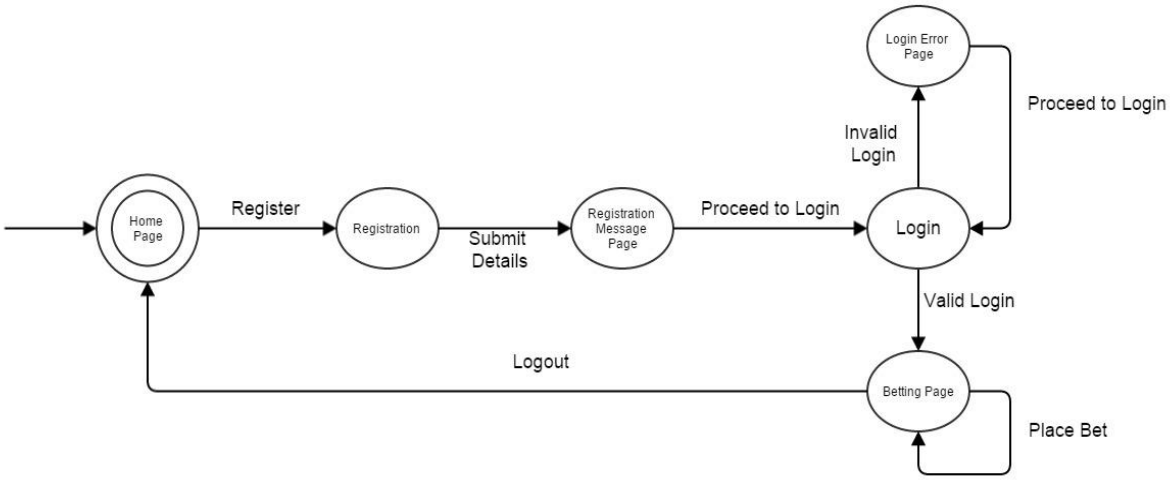


Illustration 3: Model Design

8.3 Test Set Up

The performance test was carried out using multiple threads, each running an instance of the model. Concurrent execution of the model is guaranteed by first opening all the required browsers before executing any model. A series of tests where executed with both a Firefox driver and an HTML Unit driver, since the HTML Unit driver gives a more accurate result. This is due to the extra overheads that the Firefox driver incurs, such as memory and the time taken to switch from one browser to the other, which are not relevant to the test since the aim of the test is to analyze the response time of the site. The system was tested on Windows and Ubuntu, and for all tests the number of steps was 100, using an All Round Tester. The related classes used for the model are described below.

Class Name	Description
PerformanceTest	Contains the model of a typical user using the website
TestLauncher	Launches multiple instances of the model over different threads
States	An enumeration of the different states

8.4 Assumptions

- The model assumes that the user will never enter a wrong password for three consecutive times.
- When registering, registration data will always be correct.

8.5 Average Response Time

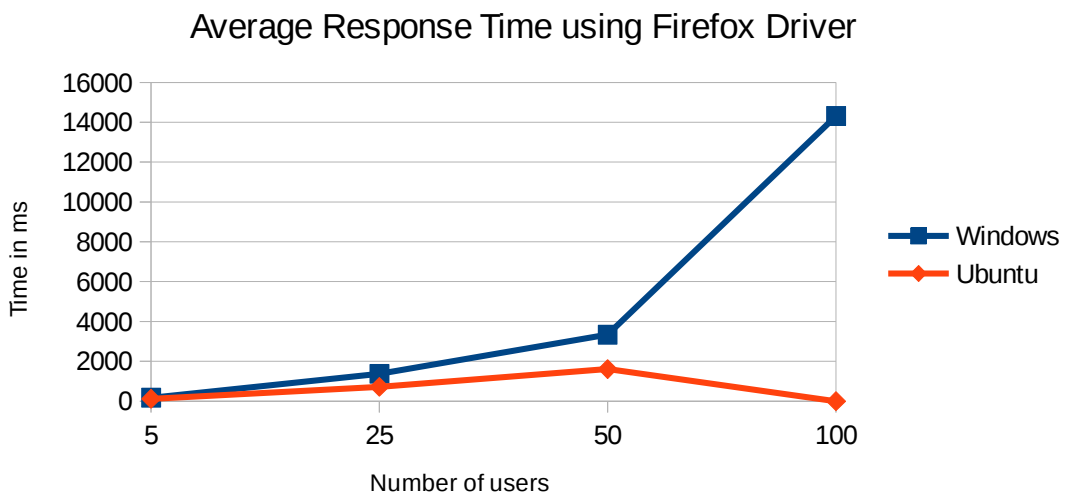
8.5.1 Time Calculation Process

The response time to load a page is calculated by calculating the difference in time from just before a navigation occurs to just after the page loads. To make sure that the page has loaded, the current URL is asserted asserted against the expected URL. The differences calculate are added to a vector, which is shared amongst all the threads. After all threads terminate, the average response time is calculated by dividing the sum of the differences by the size of the vector (number of entries).

8.5.2 Results

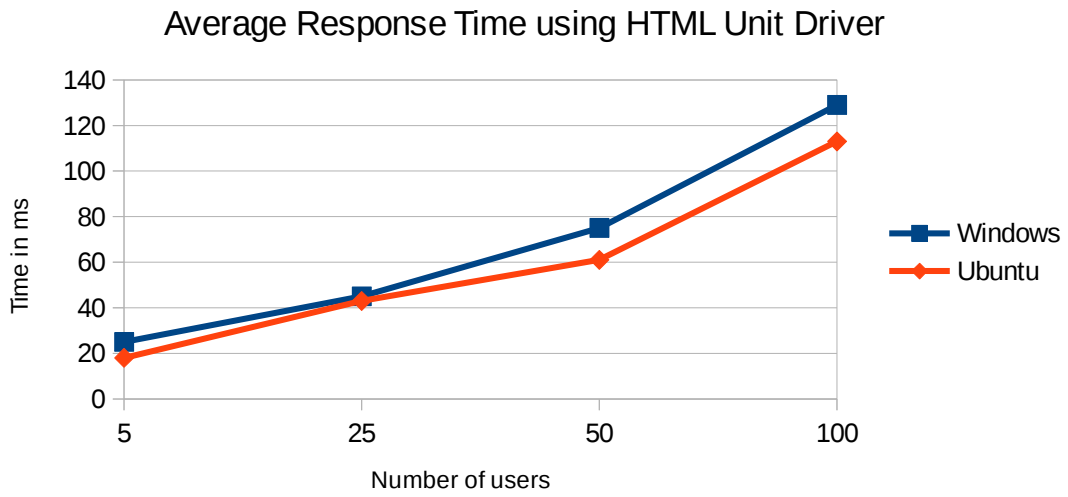
Average response time when using a Firefox Driver

Users	Windows	Ubuntu
5	176	120
25	1375	719
50	3338	1615
100	14309	Out of memory



Results when using an HTML Unit Driver

Users	Windows avg resp time	Ubuntu
5	25	18
25	45	43
50	75	61
100	129	113



From the results provided above, one can notice that the average response time of the site depends on the number of concurrent users. The average response times when using an HTML Unit Driver increases linearly with the number of users, as shown by the graph above.

However, the result is not the same for the Firefox Driver, where the response time increased linearly up till 50 users, then increased significantly for 100 users. This is because 100 browsers do not fit in memory, hence additional time was spent swapping these processes back and forth to virtual memory. In fact, Ubuntu ran out of memory during this test. During these tests it was noticed that Ubuntu performs slightly better than Windows.