ANALYSIS OF QUALITY ASSURANCE ON SISTEM INFORMASI ZAKAT (SIZAKAT) THROUGH SOFTWARE TESTING

Abdul Haris, Wisnu Jatmiko, and Ari Wibisono

Fakultas Ilmu Komputer, Universitas Indonesia, Kampus Baru UI Depok, Jawa Barat, 16424, Indonesia

E-mail: abdul.haris91@ui.ac.id

Abstract

Sistem Informasi Zakat (SIZakat) is a web-based information system that is used to assist in the management of zakat in Imam Bonjol Mosque Pondok Labu, South Jakarta. In this thesis, we conducted testing to the SIZakat application to know the quality and the feasibility. We conducted seven kinds of testing: Unit Testing, Integration Testing, Stress Testing, Load Testing, Testing SQL Injection, XSS Injection Testing and User Acceptance Testing. In addition to ensure the quality of SIZakat, the SIZakat test result is expected to be a reference for future quality improvement. Test results show that SIZakat have accurate functionalities, good security, and good performance.

Keywords: *SIZakat, Unit Testing, Integration Testing, Stress Testing, Load Testing*

Abstrak

Sistem Informasi Zakat (SIZakat) merupakan sistem informasi berbasis web yang digunakan untuk membantu proses pengelolaan zakat di Masjid Imam Bonjol Pondok Labu Jakarta Selatan. Pada tugas akhir ini, kami melakukan pengujian (testing) terhadap aplikasi SIZakat untuk mengetahui kualitas dan kelayakan. Kami melakukan tujuh macam pengujian yaitu Unit Testing, Integration Testing, Stress Testing, Load Testing, SQL Injection Testing, XSS Injection Testing, dan User Acceptance Testing. Selain untuk menjamin kualitas SIZakat, diharapkan hasil pengujian SIZakat menjadi acuan untuk perbaikan mutu kedepannya. Hasil pengujian menunjukkan SIZakat memiliki fungsional yang akurat, keamanan yang baik, dan performance yang bagus.

Kata Kunci:*SIZakat, Unit Testing, Integration Testing, Stress Testing, Load Testing*

1. Introduction

The rapid development of information technology influences on the growing needs for software that can support organization's business processes.The more demand on the software to support the business processes, the more software is developed to help it.This makes so many variety and choices of software that can be used to complete the job. Therefore, in the process of making and designing software, developers must consider the needs and quality of the software being developed.

Sistem Informasi Zakat (SIZakat) is an application used to assist the process of management of zakat in Imam Bonjol Mosque, Pondok Labu, South Jakarta. The classic issues that also experienced by other mosques occur when approaching the day of Eid. A joyful moment for every muslim people around Imam Bonjol Mosque become a polemic issue itself because zakat. The renowned Imam Bonjol Mosque, as one of the great mosque has becomethe trust of the *muzakki* (person who pays zakat) in the neighborhood of Pondok Labu subdistrict. In terms of zakat, management inImam Bonjol Mosque is better than most of the mosques, while there are many other mosques using conventional methods such as recording through the books one after anotherzakat transactions, then recapitalize and record them manually later.

This method is very vulnerable and the possibility of mistakes is very high.It always happens every year and until now still have not found the effective solution. Would be a pity that the method continues to be used when the risk is always repeated every year. Especially Mosque Imam Bonjol itself always increase the amount of zakat almost every year. Through this program mosques and Zakat Distribution Units (UPZ) is expected to be able to manage the distribution of zakat transparently and accountably.

One of the major problems in the management of zakat in Imam Bonjol Mosque was also associated with the habits of the people around Imam Bonjol Mosque who often paid zakat when approaching D-day. At its peak, the number of zakat transactions increased rapidly. This is a problem because the distribution of zakat must be completed before the preacher climbing up the pul-

pit during the Eid prayer, otherwise it would not be counted as 'zakat'instead as an ordinary 'charity'. Whereas most people paid zakat at night the day before. SIZakat will accommodate the needs of *amilin* (the zakat manager) to predict the amount of zakat al-Fitr should be issued by *amilin*for this year based on the data in the previous year. Therefore, there needs to be a quality assurance of SIZakat in terms of performance, accuracy, and security.

Based on the estimated number of zakat transactions mentioned above, SIZakat should have good performance to serve requests from many users, a good security because reports of zakat are important documents that should be kept confidential in order to avoid errors in the input and calculation of zakat, and the functional accuracy of SIZakat also needs to be ascertained because the functions in SIZakat closely related to the distribution of reports.

Every software that will be released to the public need to go through a process of quality assurance or often called the Software Quality Assurance (SQA). SQA needs to be done to determine the quality and feasibility of the software. The process is necessary to minimize losses due to the low quality software. Nowadays, both desktop application and web application are needed to support business processes. Before it was released to the public, an application passed several stages in the process of software quality assurance where the purpose of this process can be seen from different viewpoints.

One important perspective is how to ensure and maintain the quality of the application and convince consumers that the application can be accepted in society.

2. Methodology

This paper discusses SIZakat's quality case study that will be used as a support in the management of zakat in Imam Bonjol Mosque. As the title suggests, we will conduct software testing to measure SIZakat's quality. Speaking of software testing, there must be association with software development model.Hambling, Morgan and Samaroo [1] stated there are 3 (three) models that commonly used in software development, they are waterfall model, V-Model, and Iterative Development. In V-Model, testing an application starting from unit testing, integration testing, system testing, and then acceptance testing as the final test (Figure 1). The scope of this study is to perform 7 (seven) different types of tests to determine the quality of SIZakat. The seven tests are Unit Testing, Integration Testing, Stress Testing, Load Testing, SQL Injection Testing, XSS Injection Testing and User Acceptance Testing.

Unit Testing

According to the Laudons [2], unit testing involves testing each program or code separately in the system. Shrivastava and Jain [3] say that program testing is another name for unit testing. This test is intended to ensure that the written code for a unit already meets the specifications, before integrated with other units [1]. According to Seixas, Fonseca, Vieira, and Madeira [4], a good writing and structure of code will also improve a web security. We usedSimpleTest, a unit testing framework that is open source and can be used to test the PHP programming language (Baker) and also compatible with CodeIgniter framework.SimpleTest can test whether the written code in SIZakat units can run in accordance with the specifications. With SimpleTest, we can create a test case for each class to be tested.

Integration Testing

Integrationtesting is performed to determine whether the collection of classes that must work together can run without error. The purpose of integration testing is to find damage to the interaction interfaces between components or integrated systems. Thus, the basis of test on integration testing may include: system and software design; diagram of the system architecture, workflow, and use-case. Testing can be done starting from the smallest or largest unit [5]. SeleniumIDEis selected to perform integration testing as this tool is portable, provides tool record, and playback for authoring test without learning new scripting test language [6]. Test cases that have been created are stored into many file types such as HTML, Perl, PHP, JUnit, Ruby, and others.

Stress Testing

According to Kunhua Zhu, Junhui Fu, and Yancui Li [7], stress test was done by gradually increasing

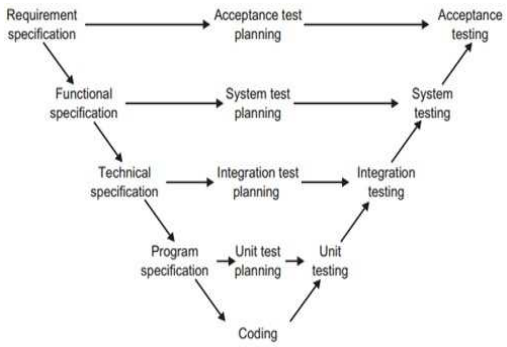

Figure 1. V-Model

the load of the system to test the changing performance of the system. Stress test examines whether the state of hardware and software system environment can withstand the maximum load and to help identify the bottleneck in the system. In this test, we used the standard testing tools used for Apache Web Server that is Apachebench (ab). This tool prints output which is very useful to determine some performance aspects of web server.

Load Testing

According to Subraya [8], load testing is used to determine whether the system being tested is able to handle anticipated activities carried out simultaneously by different users. To simulate such things in real events, we used a tool called Gatling Tools. Gatling is a testing tool that runs on top of Java Virtual Machine (JVM) using the Scala simulation script that can measure performance of client/server applications. By default, Gatling can be used to measure performance of HTTP protocol only (web application). However, users can add their desired protocol support to Gatling by themselves [9].

SQL & XSS Injection Testing

SQL and XSS Injection Testing aims to test the database security and XSS attacks (Cross-Site Scripting) in SIZakat respectively. SQL Injection ranks first in the 10 list of web application weaknesses issued by the Open Web Application Security Project (OWASP) as stated by several researchers [10-11]. To facilitate the inspection and detection of SQL Injection found in the database, weused a tool called SQL Mapper (sqlmap). This tool is developed using Python which does not rely on the operating system being used and easy to operate. We usedsqlmap because it can be used for all types of databases, operating systems and can be used to get the database name, table name even get important contents of a table from an application accurately. XSS Injection ranks second after SQL Injection in the top 10 list of web applicationweaknesses issued by OWASP [11]. To detect the presence of a loophole for XSS attacks, we used a tool called XSS-Me, a plugin for the Mozilla Firefox browser. For the moment, XSS-Me can only test reflected XSS and does not include with stored XSS [12]. Although such attack is quite dangerous, this test is enough to protect applications from XSS attacks. We used XSS-Me because it has enough features and is very easy to use.

Acceptance Testing

Acceptance testing gives the final certification that the system is ready for use on production levels [2]. According to Hambling, Morgan, and Samaroo [1], the purpose of acceptance testing is to provide users with confidence that the system will functioning in accordance with their expectations. Acceptance testing was done by evaluating the system by the users and stakeholders, and if all parties are saisfied when the system has met their standards, the system is formally accepted for installation.

3. Results and Analysis

Unit Testing

The test is performed on localhost which is located in author's computer. In this test, we examine a unit or a class or a method that exists in models. Models are PHP classes that are designed to work with the database [13]. The unitsare in models because SIZakat was developed using CodeIgniter.

To ensure each method issuing the correct output, we look at the use of the method on the controller. We look at what input is needed and the re-

TABLE 1
UNIT TESTING RESULTS 1

Model class name	Method Name	Result
mustahik_model	getAll()	PASS
	count_mustahik()	PASS
	get_mustahik_page()	PASS
	add_mustahik()	PASS
	update_mustahik()	PASS
	delete_mustahik()	PASS
	get_mustahik_by_id()	PASS
	search_mustahik()	PASS
	get_userid_by_name()	PASS
	get_photo_by_id()	PASS
muzakki_model	getAll()	PASS
	count_muzakki()	PASS
	get_muzakki_page()	PASS
	add_muzakki()	PASS
	add_muzakki_inTransaction()	PASS
	add_compact_muzakki()	PASS
	update_muzakki()	PASS
	delete_mustahik()	PASS
	get_muzakki_by_id()	PASS
	get_userid_by_name()	PASS
	get_userid_by_username()	PASS
	get_photo_by_id()	PASS
periode_model	getAll()	PASS
	get_periode_by_id()	PASS
	get_periode_by_year()	PASS
	get_periode_by_status()	PASS
	count_periode()	PASS
	get_periode_page()	PASS
	add_periode()	PASS
	update_status_periode()	PASS
	process_update_periode()	PASS
prediction_model	getAll()	PASS
	getAllYear()	PASS
	getLastYear()	PASS
	getAllSum()	PASS
	getAllSumMuzakki()	PASS

sult generated from the method. In the controller, we can also find out what methods are used and what not. It helps in saving time because we can test those methods that are used in SIZakat. After finding out the needed input for the method, we then make a statement to compare the method's output with the expected result. Suppose to examine a method to calculate the user, then the expected result with the output of the method is same, which is a number. Not only of its type, but also the amount has to be the same.

TABLE 2
UNIT TESTING RESULTS 2

Model class name	Method Name	Result
report_model	getAll()	PASS
	get_transaction_page()	PASS
	countTransc()	PASS
	get_zakat_muzakki_id_by_date()	PASS
	get_transaction_by_zakat_type_and_date2()	PASS
	get_transaction_pertanggal2()	PASS
	insert_batch_report_model()	PASS
	get_batch_report_model()	PASS
	get_batch_report_model_by_year()	PASS
user_model	count_user()	PASS
	countUserRole()	PASS
	get_user_page()	PASS
	get_all_users()	PASS
	update_user()	PASS
	delete_user()	PASS
	delete_user_by_username()	PASS
	add_user()	PASS
	get_user()	PASS
	get_name_by_id()	PASS
	get_user_by_id()	PASS
	get_role_user()	PASS
	get_photo_by_id()	PASS
zakat_quality_model	getAll()	PASS
	get_zakat_quality_by_zakatType()	PASS
	get_zakat_quality_desc_by_keys()	PASS
	get_zakat_quality_by_id()	PASS
	get_ztID_by_zqID()	PASS
	count_zakat_quality()	PASS
	get_latest_id()	PASS
	add_zakat_quality()	PASS
	process_update_zakat_quality()	PASS
	delete_zakat_quality()	PASS
	countZakatTranscbyType()	PASS
zakat_type_model	getAll()	PASS
	get_zakat_type_description_by_key()	PASS
	count_zakat_type()	PASS
	get_zakat_type_page()	PASS
dist_zakat_mustahik	getAllDistZakatMustahikTraining()	PASS
	getAllDistZakatMustahikPredict()	PASS
	insertIfNewPeriod()	PASS
	update_distribution_zakat()	PASS
	live_update_distribution_zakat()	PASS
	getDataMustahikWithZakatDist()	PASS
	getRiwayatZakatMustahik()	PASS
	delete_mustahik_distribution()	PASS

In Table 1 and 2, listed all model classes used in SIZakat.There are also methods on every model class that has successfully passed the unit testing. It can be seen from the Result column that says the success of a method. If the method has passed within expectations that have been determined, then the Result of the method is PASS otherwise the Result is FAIL meaning the results of the method do not have the same type or different amounts.

Integration Testing

TABLE 3
INTEGRATION TESTING RESULTS 1

Menu	Feature	Test Results Log
User	User Data Management	Runs: 1 Failures: 0
	Viewing User Details	Runs: 1 Failures: 0
	Adding User Data	Runs: 1 Failures: 0
	Changing User Data	Runs: 1 Failures: 0
	Deleting User Data	Runs: 1 Failures: 0
	Muzakki Data Management	Runs: 1 Failures: 0
Muzakki	Viewing *Muzakki* Details	Runs: 1 Failures: 0
	Adding *Muzakki* Data	Runs: 1 Failures: 0
	Changing *Muzakki* Data	Runs: 1 Failures: 0
	Deleting *Muzakki* Data	Runs: 1 Failures: 0
Mustahik	*Mustahik* Data Management	Runs: 1 Failures: 0
	Viewing *Mustahik* Details	Runs: 1 Failures: 0
	Adding *Mustahik* Data	Runs: 1 Failures: 0
	Changing *Mustahik* Data	Runs: 1 Failures: 0
	Deleting *Mustahik* Data	Runs: 1 Failures: 0
Zakat Quality	Zakat Quality Management	Runs: 1 Failures: 0
	Viewing Zakat Quality Details	Runs: 1 Failures: 0
	Adding Zakat Quality Da-ta	Runs: 1 Failures: 0
	Changing Zakat Quality Data	Runs: 1 Failures: 0
	Deleting Zakat Quality Data	Runs: 1 Failures: 0

The test was conducted on SIZakat running on the Faculty of Computer Science UI (Fasilkom)server with address at http://ws-73.rsa.cs.ui.ac.id/sizakat. In this test, we logged-in to system using all roles then run all existing use-cases to determine whether the function is going well and according to the scenario. In addition, it is necessary to see whether the function is also showing the expectedinterface. We used Selenium IDE 2.0.0 and Mozilla Firefox browser to perform this test. The list of use-cases that have been tested can be seen in Table 3 and 4.

A green bar expresses that the testing goes well from beginning to end, whereas a red bar expresses that an error has occurred in the test. In Table 3 and 4, it can be seen that all existing use-cases have passed the test which are marked with green bars.

TABLE 4
INTEGRATION TESTING RESULTS 2

Menu	Feature	Test Results Log
Report	Creating Customized Report	Runs: 1 Failures: 0
	Creating Batch Report	Runs: 1 Failures: 0
Prediction	Viewing Zakat Prediction Report	Runs: 1 Failures: 0
	Viewing *Muzakki* Prediction Report	Runs: 1 Failures: 0
Transaction	Zakat Transaction Data Management	Runs: 1 Failures: 0
	Viewing Transaction Details	Runs: 1 Failures: 0
	Changing Zakat Transaction	Runs: 1 Failures: 0
	Removing Zakat Transaction	Runs: 1 Failures: 0
	Adding Zakat Transactions	Runs: 1 Failures: 0
	Muzakki Transaction Data Management	Runs: 1 Failures: 0
	Viewing Transaction History	Runs: 1 Failures: 0
Period	Period Management	Runs: 1 Failures: 0
	Adding Period	Runs: 1 Failures: 0
	Changing Period	Runs: 1 Failures: 0
General Functions	Login	Runs: 1 Failures: 0
	Logout	Runs: 1 Failures: 0

Stress Testing

In the analysis of this test, we consider four parameters to form the basis to determine web performance. The four parameters are complete requests, failed requests, requests per second, and transfer rate. Out of the four parameters, the complete requests and failed requests parameters are interconnected. The complete requests value is the amount of overall requests reduced by the number of failed requests, and vice versa.

The common notations used for testing is -n (number of requests or the number of users) and -c (number of concurrent users) [14]. The -c notation is used to perform stress testing, a test aimed to determine performance of the application when accessed simultaneously. For example, we want to test anapplication with address at http://ws-73.rsa.cs.ui.ac.id. We would like to know performance of the application when it accessed by 100 people and 10 of them simultaneously accessed it. So the used notation is "`ab -n 100 -c 10 http://ws-73.rsa.cs.ui.ac.id`". This test will generate some important parameters that show information from the test performed. Example outputs generated from this trial are: the number of complete requests is 100, the number of failed requests is 0, the number of requests per second is 57.87, and the number of the transfer rate is 303.41. From these examples, the number of complete requests equals to the number of users were tested which is 100.

To determine performance of SIZakat, we used the four parameters mentioned earlier. We specify the criteria or limits of the four parameters to determine performance of SIZakat. If the value of the four parameters included in the criteria then SIZakat have a good performance. Below we will explain the criteria of each parameter:

Complete request
Complete request is the number of successful requests or responses received. The number of complete requests must be in accordance with the number of users tested.

Failed request
Failed request is the number of which is considered failed to be received by a user. If the the value of failed requests is greater than zero, there will be printed on the other line showing number of requests that failed because of the connection, readability, wrong data size, or exceptions. For testing on SIZakat, we determine that the value of failed requests should be no more than zero (0).

TABLE 5
STRESS TESTING RESULTS OF 500 USERS

Concurrence Level	Notation	Hasil Pengujian: Complete Requests	Failed Requests	Request per Second [#sec] (mean)	Transfer Rate [Kbytes/sec] received
100	Ab –n 500 –c 100	500	0	58.99	309.31
200	Ab –n 500 –c 200	500	55	39.84	192.83
300	Ab –n 500 –c 300	500	139	55.99	236.63
400	Ab –n 500 –c 400	500	110	53.52	236.66
500	Ab –n 500 –c 500	500	89	53.40	245.20

TABLE 6
STRESS TESTING RESULTS OF 1000 USERS

Concurrence Level	Notation	Hasil Pengujian: Complete Requests	Failed Requests	Request per Second [#sec] (mean)	Transfer Rate [Kbytes/sec] received
100	Ab –n 500 –c 100	1000	0	58.18	305.05
200	Ab –n 500 –c 200	1000	165	61.43	285.83
300	Ab –n 500 –c 300	1000	260	57.22	236.63
400	Ab –n 500 –c 400	1000	221	52.18	236.66
500	Ab –n 500 –c 500	1000	89	53.40	245.20

Requests per second
Requests per second are the number of requests that is able to be served in one second. The greater the value of requests per second the better. This parameter displays the value of the average number of requests that can be served in one second. For testing on SIZakat, we determine that on average more than 10 requests/second is a good result.

Transfer rate
Transfer rate is a parameter that indicates the capacity of data that can be displayed. The greater the value of this parameter, the better performance SIZakat has. A good value for this parameter is more than 10 Kbyte.

In this test, we tested SIZakat which is already installed on the Fasilkomserver. The results of the test which performed directly on the Fasilkom server generates output that is more accurate and shows the true state. We will explain the analysis of test results based on the number of users increasing over time.

In the first stress test, we used500 users with 100 concurrent users increased on each subtest, while in the second stress testingwe used 1000 users with 100 concurrent users increased on each subtest, but only limit it to 500. From Table 5 and 6, we conclude:1) The number of complete requests is equal to the number of users, 2) The number of failed requests for concurrence level of 200-500 is greater than zero. Only at concurrence level of 100 is zero, 3) The number of requess per second for all concurrence levels is more than 10 requests per second, and 4) The number of transfer rate for all concurrence levelsis more than 10 Kbytes/sec.

Load Testing

There are two (2) variables and three (3) parameters used to perform this test. The first variable is the number of users who accessed SIZakat and second is the ramp period allocated for testing. For example, the number of users is 100 and the ramp period (in sec) is 2 so 100 users who make requests are served within 2 seconds or equal to 50 requests per second. The test results are presented in tabular form which can be found in Table 7. Furthermore, from the results of the testwe process the data to get the parameters: min, max, and mean response times from Global Information, the overall statistic request. According to Mizouni, Serhani, Dssouli, Benharref, and Taleb [15] response time is the time required between issuing a request and getting the response. Those three parameters of time determine performance of SIZakat. The time unit for each response time is millisecond.

To determine performance of SIZakat, we only consider the Time Average which is the average time spent to serve concurrent requests. A good response time is 10 seconds [8].The testing results of entire menus of SIZakatcan be seen in Table 7 using 100 users and 5 seconds of ramp period or equal to 20 requests per second.

SQL Injection Testing

SQL injection testing was carried outon SIZakat that located on a Fasilkom server with address at http://ws-73.rsa.cs.ui.ac.id/sizakat. There are two ways to execute SQL Injection.They are to try some unnatural characters forcibly (brute force) and using dorks [16].

TABLE 7
LOAD TESTING RESULTS

Menu	Functional	Time Average (ms)	Time Interval (ms) Min	Max
User	User Data Management	1273	50	2980
	Viewing User Details	954	40	5560
	Adding User Data	179	40	1080
	Changing User Data	311	70	3680
Muzakki	*Muzakki* Data Management	810	50	1770
	Viewing *Muzakki* Details	1703	50	4660
	Adding *Muzakki* Data	121	50	310
	Changing *Muzakki* Data	266	90	690
Zakat Quality	Zakat Quality Management	160	40	510
	Viewing Zakat Quality Details	105	50	360
	Adding Zakat Quality Data	130	40	250
	Changing Zakat Quality Data	318	70	1370
Report	Creating Batch Report	219	80	710
Prediction	Viewing Zakat Prediction Report	822	50	2290
	Viewing *Muzakki* Prediction Report	507	40	1150
Transaction	Zakat Transaction Data Management	881	50	1600
	Viewing Transaction Details	1183	60	8230
	Changing Zakat Transaction	1137	220	5480
	Adding Zakat Transactions	1410	90	4880
	Muzakki Transaction Data Management	406	80	790
	Viewing Transaction History	288	60	710
Period	Period Management	131	40	550
	Adding Period	369	40	2360
	Changing Period	179	90	350
	Activating Period	192	70	410
	Deactivating Period	191	60	480
General	Login	244	40	750
	Logout	288	30	820

We used the second injection technique which means by using a dork. This technique is usually used when a website has a dork that can be tried to find errors in the database. SIZakat is different from other web applicationsin institution or organization websites as they are more informative. Usually on institution or organization websites, many dorks can be found that can be used to perform SQL Injection. SIZakat is an application where its role has been determined. Unauthorized users can only access SIZakat up to the loginpage. Only users who have been registeredthat can find SIZakat's dorks. Although dorks in SIZakat have been found, the dorks are not necessarily can be used to perform SQL Injection. Example of dorks in SIZakat:
/manage_user/view_user/STF201208081
/transaction/detail_transaction/TRANSC2013012340

From the dorks above, these can be tried to find errors in SIZakat database. The test is performed by adding a single quote "'" after id and minus "-" before the id in the URL address. Wedidn't get an error when adding those two signs in SIZakat. In other words, SIZakat security can not be penetrated via SQL Injection with this simple step. If there is an error message such as "*You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near ''1'' at line 1*", then the process of SQL Injectiontesting can be continued.

To support SQL Injection testing we used sqlmap toolwith version 1.0-dev. Thistool scans all vulnerabilities that can be used for SQL Injection in SIZakat.

The next test was done by using the dork addresses in SIZakat automatically. We executed a query in Figure 2 and got the result shown in Figure 3.

```
sqlmap -u http://ws-
73.rsa.cs.ui.ac.id/sizakat/index.php/manage
_user/view_user/STF201208081
```

Figure 2. Sqlmap Query

```
[10:37:30] [INFO] testing connection to the
target URL
[10:37:30] [INFO] testing if the target URL
is stable. This can take a couple of
seconds
[10:37:31] [INFO] target URL is stable
[10:37:31] [CRITICAL] no parameter(s) found
for testing in the provided data (e.g. GET
parameter 'id' in
'www.site.com/index.php?id=1')
```

Figure 3. Sqlmap Result

```
sqlmap -u http://ws-
73.rsa.cs.ui.ac.id/sizakat/index.php/manage
_user/view_user/STF201208081*
```

Figure 4. Sqlmap Query

```
[10:46:32] [INFO] testing connection to the
target URL
[10:46:32] [INFO] testing if the target URL
is stable. This can take a couple of
seconds
[10:46:33] [INFO] target URL is stable
[10:46:33] [INFO] testing if URI parameter
'#1*' is dynamic
[10:46:33] [WARNING] URI parameter '#1*'
does not appear dynamic
[10:46:33] [WARNING] heuristic (basic) test
shows that URI parameter '#1*' might not be
injectable
[10:46:33] [INFO] testing for SQL injection
on URI parameter '#1*'
[10:46:44] [WARNING] using unescaped
version of the test because of zero
knowledge of the back-end DBMS. You can try
to explicitly set it using option '--dbms'
[10:46:48] [WARNING] URI parameter '#1*' is
not injectable
[10:46:48] [CRITICAL] all tested parameters
appear to be not injectable. Try toincrease
'--level'/'--risk' values to perform more
tests. Also, you can try to rerun by
providing either a valid value for option
'--string' (or '--regexp')
```

Figure 5. Sqlmap Result

```
[10:46:32] [INFO] testing connection to the
target URL
[10:46:32] [INFO] testing if the target URL
is stable. This can take a couple of
seconds
[10:46:33] [INFO] target URL is stable
[10:46:33] [INFO] testing if URI parameter
'#1*' is dynamic
[10:46:33] [WARNING] URI parameter '#1*'
does not appear dynamic
[10:46:33] [WARNING] heuristic (basic) test
shows that URI parameter '#1*' might not be
injectable
[10:46:33] [INFO] testing for SQL injection
on URI parameter '#1*'
[10:46:44] [WARNING] using unescaped
version of the test because of zero
knowledge of the back-end DBMS. You can try
to explicitly set it using option '--dbms'
[10:46:48] [WARNING] URI parameter '#1*' is
not injectable
[10:46:48] [CRITICAL] all tested parameters
appear to be not injectable. Try toincrease
'--level'/'--risk' values to perform more
tests. Also, you can try to rerun by
providing either a valid value for option
'--string' (or '--regexp')
```

Figure 6. Sqlmap Result

```
sqlmap –u "http://ws-
73.rsa.cs.ui.ac.id/sizakat/index.php/manage
_user/view_user/STF20120881*" --dump
```

Figure 7. Sqlmap Query.

```
[17:07:29] [INFO] testing connection to the
target URL
[17:07:29] [INFO] testing if the target URL
is stable. This can take a couple of
seconds
[17:07:37] [INFO] target URL is stable
[17:07:37] [INFO] testing if URI parameter
'#1*' is dynamic
[17:07:38] [INFO] confirming that URI
parameter '#1*' is dynamic
[17:07:38] [INFO] URI parameter '#1*' is
dynamic
[17:07:38] [WARNING] heuristic (basic) test
shows that URI parameter '#1*' might not be
injectable
[17:07:38] [INFO] testing for SQL injection
on URI parameter '#1*'
[17:07:46] [WARNING] using unescaped
version of the test because of zero
knowledge of the back-end DBMS. You can try
to explicitly set it using option '--dbms'
[17:07:49] [WARNING] URI parameter '#1*' is
not injectable
[17:07:49] [CRITICAL] all tested parameters
appear to be not injectable. Try to
increase '--level'/'--risk' values to
perform more tests. Also, you can try to
rerun by providing either a valid value for
option '--string' (or '--regexp')
```

Figure 8. Sqlmap Result

From the scanning result above, sqlmap can not perform the test because it only supports query-string-based URL. For that we need a special command to test more focused on ID. We executed a query in Figure 4 and got the result in Figure 5.

Then the second test on dork address at "/transaction/detail_transaction/TRANSC2013012340" got the report as seen in Figure 6.

For the final test we tried to retrieve tables, users, and passwords that exist in the database. We executed a query shown in Figure 7 and got the result seen in Figure 8.From both completed tests, we conclude that sqlmap can not penetrate the database security in SIZakat.

XSS Injection Testing

The last security testing is XSS Injection testing. The XSS Injection testing was carried outon SIZakat that located on a Fasilkom server with address at http://ws-73.rsa.cs.ui.ac.id/sizakat.

To support XSS Injection technique we used XSS Me with version 0.4.6. This tool performs brute-force attacks against the forms on SIZakat webpage so it can find a vulnerability that can be used for XSS Injection. The testwas carried out by using the period changing menu in SIZakat. In Table 8. we can see the results of test on webpage with "*Test all forms with top attacks*".

TABLE 8
XSS INJECTION TESTING RESULTS

Tested Value	Result
<SCRIPT <B>document.vulnerable=true;</SCRIPT> <<SCRIPT>document.vulnerable=true;//<<</SCRIPT> <BODY onload!#$%&()*~+-_.,:;?@[/\]^`=document.vulnerable=true;> <IMG SRC="javascript:document.vulnerable=true;"> <IMG SRC=" &#14; javascript:document.vulnerable=true;"> <IMG SRC="jav ascript:document.vulnerable=true;"> <SCRIPT>document.vulnerable=true;</SCRIPT> <META HTTP-EQUIV="Set-Cookie" Content="USERID=<SCRIPT>document.vulnerable=true</SCRIPT>"> <meta http-equiv="refresh" content="0;url=javascript:document.vulnerable=true;">	The unencoded attack string was not found in the html of the document. DOM was not modified by attack string. Field does not appear vulnerable to XSS String

After conducted 18 types of XSS attacks using XSS Me, the injected script code can not be found in SIZakat webpages that have been tested.The message "*The unencoded attack string was not found in the html of the document*" which states that the attack code was not found on webpage indicates that SIZakat can not be injected.

User Acceptance Testing

User Acceptance Testing (UAT) is a test conducted by SIZakat userrepresentatives to check that if the system has been developed to meet their needs. This test is a part of Factory Acceptance Testing (FAT) where the system is tested by the user before it moved to the user's location.

In this test, we will utilize a UAT document which handedto SIZakat's users. This document contains a list of scenarios to be tested by the user, along with instructions on how to complete the scenarios and desired outcome of the scenarios. The scenariosused in this test are use-caseswhich are from client's requirements.

This test was done by user doing all use-case that is available as instructed. When a use-case has been completed and the system appropriately displays what has been said in the UAT document, that use-case passes the test, and then user creates a checkmark in the result column of the use-case. This test was done by 2 users and the result is all use-case got a checkmark (Table 9) which indicates that all SIZakat use-cases are consistent with the specifications.

4. Conclusions

This study has resulted in a test results document that can be used to consider whether or not SIZakat is fit for use. The following conclusions were obtained by doing allperformed tests.

TABLE 9
USER ACCEPTANCE TESTING RESULTS

No.	Use-case	Result
1	User Data Management	✓
	Viewing User Details	✓
	Adding User Data	✓
	Changing User Data	✓
	Deleting User Data	✓
2	*Muzakki* Data Management	✓
	Viewing *Muzakki* Details	✓
	Adding *Muzakki* Data	✓
	Changing *Muzakki* Data	✓
	Deleting *Muzakki* Data	✓
3	*Mustahik* Data Management	✓
	Viewing *Mustahik* Details	✓
	Adding *Mustahik* Data	✓
	Changing *Mustahik* Data	✓
	Deleting *Mustahik* Data	✓
4	Zakat Quality Management	✓
	Viewing Zakat Quality Details	✓
	Adding Zakat Quality Data	✓
	Changing Zakat Quality Data	✓
	Deleting Zakat Quality Data	✓
5	Creating Customized Report	✓
	Creating Batch Report	✓
6	Viewing Zakat Prediction Report	✓
	Viewing *Muzakki* Prediction Report	✓
7	Zakat Transaction Data Management	✓
	Viewing Transaction Details	✓
	Changing Zakat Transaction	✓
	Removing Zakat Transaction	✓
	Adding Zakat Transactions	✓
	Muzakki Transaction Data Management	✓
	Viewing Transaction History	✓
8	Period Management	✓
	Adding Period	✓
	Changing Period	✓
9	Login	✓
	Logout	✓

The results of unit testing showed satisfactory results because each class and method in SIZakat meets the criterias.It can be seen from all test cases that have passed the test for having produced the correct and consistent with those expected.

The integration test results showed that all functionals have been running well according to their functions. The reports from Selenium IDE indicate that every step in all scenarios have been run well when doing playback and found no errors on the interfaces.

The stress testing results indicate that the performance is good enough when SIZakat faced abnormal load. When tested using 500 and 1000 requests, SIZakat is able to serve concurrency level of 100 without fail. Judging from SIZakat location usage, this request amount is sufficient for daily needs.

The load testing results indicate that the performance is good enough for SIZakat when facing various kinds of activity from user when accessed simultaneously. The report from Gatling tool indicates that the average response time spent by the

user for each activity is no more than the time specified, which is 10 seconds.

SIZakat can not be injected using SQL Injection technique either manually or with the help of sqlmap tool. Testing by using sqlmap indicates SIZakat can not be injected because it didn't show important information about the database. SIZakat uses CodeIgniter framework that separates between the model, view, controller (MVC). In general, applications that use MVC model are safe from SQL Injection techniques.

SIZakat can not be injected using XSS Injection techniques either manually or with the help of XSS-Me tool. Either testing manually or using the XSS Me tool indicates that SIZakat can not be injected because it has the ability to validate user input.

The conclusion of all testing results is SIZakat already can be used to manage zakat. The conclusion from all testing results are SIZakat already can be used to manage zakat. However,it needs to do bit of repair and modification.

References

[1] Hambling, B., Morgan, P., & Samaroo, A. (2010). *Software Testing: An ISTQB-ISEB Foundation Guide* (2nd ed.). Swindon: British Computer Society.

[2] Laudon, K. C., & Laudon, J. C. (2011). *Management Information Systems* (12th ed.). New Jersey: Prentice Hall.

[3] Shrivastava, D. P., & Jain, R. C. (2011). Unit test case design metrics in test driven development. *International Conference on Communications, Computing and Control Applications* (CCCA), 1-6.

[4] Seixas, N., Fonseca, J., Vieira, M., & Madeira, H. (2009). Looking at Web Security Vulnerabilities from the Programming Language Perspective: A Field Study. *International Symposium on Software Reliability Engineering (ISSRE)*, 129-135.

[5] Craig, R. D., & Jaskiel, S. P. (2002). *Systematic Software Testing.* Massachusetts: Artech House Publishers.

[6] Selenium. (2006-2013). *Selenium IDE Plugins*. Retrieved May 28, 2013, from http://docs.seleniumhq.org/projects/ide/.

[7] Kunhua Zhu, Junhui Fu, & Yancui Li. (2010). Research the performance testing and performance improvement strategy in web application. *2nd International Conference on Education Technology and Computer (ICETC)*, 2, 328-332.

[8] Subraya, B. (2006). *Integrated Approach to Web Performance Testing: A Practitioner's Guide*. Pennsylvania: Idea Group Inc.

[9] Gatling Project. (2013). *Stress Tool*. Retrieved May 20, 2013, from Gatling Project: http://gatling-tool.org/.

[10] Atashzar, H., Torkaman, A., Bahrololum, M., & Tadayon, M. (2011). A Survey on Web Application Vulnerabilities and Countermeasures. *International Conference on Computer Sciences and Convergence Information Technology (ICCIT)*, 647-652.

[11] You Yu, Yuanyuan Yang, Jian Gu, & Liang Shen. (2011). Analysis and Suggestions for the Security of Web. *International Conference on Computer Science and Network Technology (ICCSNT)*, 1, 236-240. Retrieved June 1, 2013.

[12] Security Compass. (2013). *XSS-Me*. Retrieved May 27, 2013, from Security Compass Labs: http://labs.securitycompass.com/exploit-me/xss-me/.

[13] EllisLab, Inc. (2013, May 21). *Models*. Retrieved from CodeIgniter User Guide: http://ellislab.com/codeigniter/user-guide/general/models.html.

[14] The Apache Software Foundation. (2013). *ab - Apache HTTP server benchmarking tool*. Retrieved April 29, 2013, from Apache HTTP Server: http://httpd.apache.org/docs/2.2/programs/ab.html.

[15] Mizouni, R., Serhani, M. A., Dssouli, R., Benharref, A., & Taleb, I. (2011). Performance Evaluation of Mobile Web Services. *IEEE European Conference on Web Services (ECOWS)*, 9, 184-191.

[16] Setiadi, A. (2011). *Penjaminan Mutu Sistem Informasi Bantuan Operasioal Sekolah melalui Pengujian Performansi Keamanan dan Keakuratan*. Universitas Indonesia. Depok: Fasilkom UI.