Integration of REST-Based Web Service and Browser Extension for Instagram Spam Detection

Antonius Rachmat Chrismanto[1], Willy Sudiarto Raharjo[2], Yuan Lukito[3]
Program Studi Informatika, Fakultas Teknologi Informasi
Universitas Kristen Duta Wacana
Yogyakarta, Indonesia

***Abstract*—In this paper, a REST-based Web Service developed in previous work was integrated with a newly developed browser extension that works in modern browser (Firefox and Google Chrome) using Greasemonkey. It uses previous collected datasets which comprised of 17.000 postings and comments from 10 Indonesian actresses whom followers are more than 10 million on Instagram. The performance of the developed web services has been evaluated and the average response time is 1678.133ms using AWS platform located in Ohio (US East 2). The proposed work is working as expected and in accuracy test, it has reached 63.125% in overall, 72% for non-stemmed data and 70% for stemmed data using 1000 test data with a processing time needed for classification is under 2s. The new extension works in Firefox and Chrome and it can utilize the web services to classify spam comments in Instagram.**

Keywords—Instagram; spam comments; REST service; web service testing; browser extension

I. Introduction

Social media is no longer just a mean for sharing information along relatives and colleagues, but it has transformed into a bigger scope and touching every aspect of human life. Social media is already used in many situations, like emergency situation [1], traveling [2], and health [3]. However, it comes with a price. According to [4], [5], and [6] there are a lot of spam comments in media social, such as YouTube, Facebook, Twitter, and Instagram. These spammers may cause some information misleading, mixed information, wasting valuable network resources, and decreasing the quality of online social networking sites [7], [8], and [9].

Nowadays, most people are using Instagram because of its characteristic of being an image-based social media. A picture speaks for thousand words by nature. According to [10], Instagram has reached 1 billion monthly users in June 2018, a significant raise from 800 million in September 2017. It shows that Instagram is gaining a huge popularity among many people, including Indonesian actress who proactively engaged with their fans to help them gain more popularity and brings more business opportunities for them.

Instagram is gradually introducing new features as posted in their press web sites (https://instagram-press.com/), but rarely seen a posting about spam detection. One of the reasons is that because spam may come in many ways and sometimes it's context-based, so it's hard to find a good balance for creating an algorithm that can detect spam comments nowadays, especially in Indonesian language. There is no implemented solution for automated Indonesian language spam detection in Instagram yet. Many previous work [11], [6], [12] used Instagram data for spam detection, but so far, there are no real implemented solution for spam detection. The research done so far was more focused on testing the accuracy of each model. Especially on Indonesian-based language, which according to [13] is still considered as one of the resource-poor languages.

In this paper, an implemented solution for automated Indonesian language spam detection is proposed by building an integration between a REST-based web service and a browser extension that can be used to detect Instagram spam comments in Indonesian language. This research contributes in enriching Indonesian language related researches and creates a ready to use Instagram spam detector. Browser extension is the option we chose since it allows us to interact with the content on Instagram without breaking same-origin policy [14].

II. Related Work

Hardinata and Tirtawangsa [11] developed spam detector in Indonesian Twitter trending topics. The spam detector works by detecting spam that utilized trending topics hashtags. The spam detection process involved human input that collected using monster game interface. Zhang and Sun [15] has published their work on a model to decrease number of spam posts in Instagram, but only applicable for English language. Ali and Okiriza [12] published their work on detecting spam comments on Indonesia's Instagram post using three different algorithms: Naïve Bayes, SVM, and XGBoost. They concluded that SVM and XGBoost got the best scores of 0.9601 and 0.9512. In all the researches, not a single of them proposed a real implemented and practical solution since they all are focusing on the accuracy of the models being tested.

This work was started in 2017 by building Indonesian spam comments detector using Naïve Bayes [16] and collected more than 25.000 postings and comments from Indonesian actress with more than 10 million followers. After data cleansing process, the final data used are 17.000 postings. From this datasets, some experiments were conducted using different algorithms and it was concluded that K-Nearest Neighbors (k-NN) gave the best results with 88.4% of accuracy [17], followed by Support Vector Machine with 78.5% [18], and Naïve Bayes with 75.5% [16].

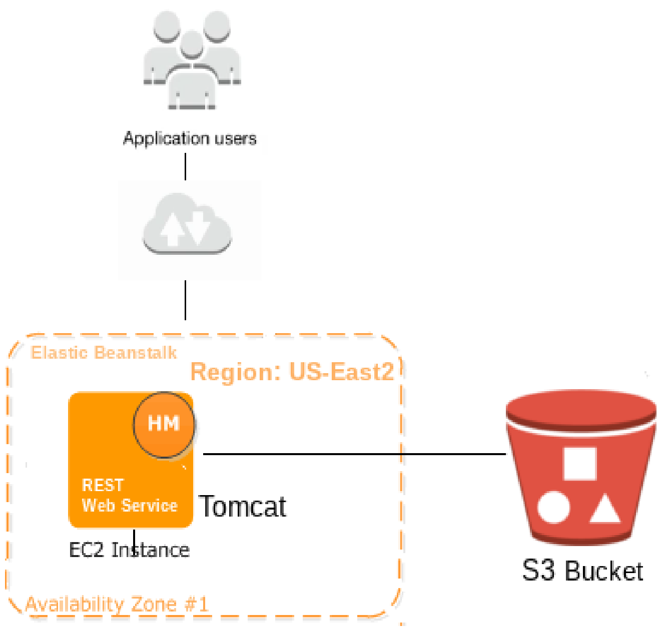


Fig. 1. Web Service Architecture.

Next, a REST-based web service to detect Indonesian language Instagram spam comments using k-NN algorithm design was designed and deployed on top of AWS platform and evaluate the performance based on response time. [19].

III. METHODOLOGY

A. Architecture

This work is using the same AWS architecture that was developed in previous work [19] for the web service architecture, which was deployed on US-East 2 region (Ohio). The system is using Tomcat as the main web server and all datasets are stored in the S3 bucket for durability and performance reason. The architecture is illustrated in Fig. 1.

The web service does not deploy SSL certificate for this machine as there are no confidential data that are communicated, and the system never stored any data transmitted to the server during spam detection process. The dataset is stored in the S3 bucket which is only accessible via the web server and not directly accessible for public.

All the communication between client (browser) and the server will be done using REST [20] which has some advantages over SOAP such as better throughput and response time, as demonstrated on [21] and [22].

B. Algorithm

In this work, k-NN algorithm is used based on previous work [17] that gives best results compared to other algorithms (Support Vector Machine [18] and Naïve Bayes [16]). K-NN is learning directly while performing classification process by finding some adjacent data object or patterns based on the input and choose a class with the highest number of patterns [19]. K-NN can be implemented as follows (Fig. 2):

1) Load the data

2) Initialize the value of k

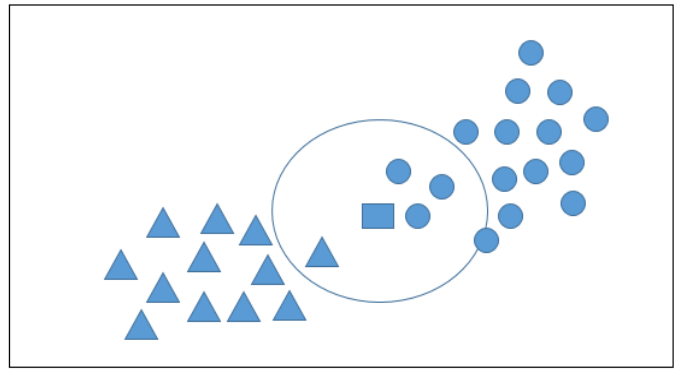


Fig. 2. k-NN Algorithm

3) For getting the predicted class, iterate from 1 to total number of training data points:

a) Calculate the distance between test data and each row of training data. Here we will use Euclidean distance as our distance metric since it's the most popular method. The other metrics that can be used are Chebyshev, cosine, etc.

b) Sort the calculated distances in ascending order based on distance values

c) Get top k rows from the sorted array

d) Get the most frequent class of these rows

e) Return the predicted class

C. Browser Extension

The browser extension is developed using Greasemonkey and works as follows:

- Script will check visited page. If it is coming from Instagram, it will add a new entry in the browser's context menu (accessed via right click)
- When user highlighted some text in Instagram posting, the extension will read the highlighted text and send it to the web service in AWS
- Web service will process the request and reply the results back to the browser
- Browser extension will display the results to user in form of a dialog box.

D. Evaluation

Several tests were conducted to evaluate some metrics. The first test was performed using SOAPUI tool which is used to perform load testing, method testing, simple load testing, burst load testing, thread load testing, variance load testing, and data-driven testing. It used 160 data for data-driven test.

The second test was testing the web service accuracy by using PHP scripts to automate the test. The test used 1000 random data taken from dataset using shuffled sampling. The dataset was generated using 10 smaller dataset which consisted of 100 data to reduce the slow processing time. Afterwards, it's merged with the rest. Next, the dataset is tested against 8 test datasets which have been stored in the web service already. The parameters used for the k-NN validation shown in Table 1.

TABLE I. TESTING PARAMETERS

Parameters	Values
Number of data	1000
Output criteria	Accuracy
Sampling type	Shuffled Sampling
Dataset type	8 types
Dataset Criteria	Unbalanced non-stemmed, unbalanced-stemmed, balanced non-stemmed, and balanced stemmed

The 8 datasets that were used are as follow:

- Generated using PHP
  - Unbalanced non-stemmed data
  - Unbalanced stemmed data
  - Balanced non-stemmed data
  - Balanced stemmed data
- Generated using R
  - Unbalanced non-stemmed data
  - Unbalanced stemmed data
  - Balanced non-stemmed data
  - Balanced stemmed data

IV. RESULTS AND DISCUSSIONS

A. Web Service Accuracy

The results of the web service accuracy after tested against 8 datasets can be seen on the Table II through Table IX.

TABLE II. DATASET 1: UNBALANCED-NON STEMED DATA GENERATED USING PHP

TEST DATA	STEM		%	NON-STEM		%
	T	***F***		***T***	***F***	
1-100	71	29	71%	73	27	73%
101-200	68	32	68%	68	32	68%
201-300	83	17	83%	83	17	83%
301-400	85	15	85%	85	15	85%
401-500	74	26	74%	73	27	73%
501-600	72	28	72%	72	28	72%
601-700	61	39	61%	61	39	61%
701-800	87	13	87%	87	13	87%
801-900	75	25	75%	75	25	75%
901-1000	58	42	58%	57	43	57%
			73%			73%

TABLE III. DATASET 2: UNBALANCED STEMMED DATA GENERATED USING PHP

TEST DATA	STEM		%	NON-STEM		%
	T	***F***		***T***	***F***	
1-100	66	34	66%	67	33	67%
101-200	53	47	53%	54	46	54%
201-300	49	51	49%	48	52	48%
301-400	44	56	44%	46	54	46%
401-500	43	57	43%	43	57	43%
501-600	58	42	58%	54	46	54%
601-700	44	56	44%	38	62	38%
701-800	55	45	55%	55	45	55%
801-900	45	55	45%	45	55	45%
901-1000	53	47	53%	55	45	55%
			51%			51%

TABLE IV. DATASET 3: BALANCED NON-STEMMED DATA GENERATED USING PHP

TEST DATA	STEM		%	NON-STEM		%
	T	***F***		***T***	***F***	
1-100	71	29	71%	73	27	73%
101-200	68	32	68%	66	34	66%
201-300	76	24	76%	75	25	75%
301-400	82	18	82%	83	17	83%
401-500	69	31	69%	69	31	69%
501-600	71	29	71%	72	28	72%
601-700	60	40	60%	60	40	60%
701-800	88	12	88%	88	12	88%
801-900	88	12	88%	88	12	88%
901-1000	65	35	65%	62	38	62%
			74%			74%

TABLE V. DATASET 4: BALANCED STEMMED DATA GENERATED USING PHP

TEST DATA	STEM		%	NON-STEM		%
	T	***F***		***T***	***F***	
1-100	69	31	69%	69	31	69%
101-200	55	45	55%	53	47	53%
201-300	51	49	51%	47	53	43%
301-400	48	52	48%	43	57	43%
401-500	50	50	50%	40	60	40%
501-600	48	52	48%	45	55	45%
601-700	62	38	62%	58	42	58%
701-800	53	47	53%	47	53	47%
801-900	48	52	48%	45	55	45%
901-1000	55	45	55%	51	49	51%
			54%			49%

TABLE VI. DATASET 5: UNBALANCED NON-STEMMED DATA GENERATED USING R

TEST DATA	STEM		%	NON-STEM		%
	T	***F***		***T***	***F***	
1-100	86	14	86%	86	14	86%
101-200	78	22	78%	77	23	78%
201-300	81	19	81%	84	16	84%
301-400	89	11	89%	86	14	86%
401-500	83	17	83%	83	17	83%
501-600	81	19	81%	77	23	77%
601-700	79	21	79%	79	21	79%
701-800	83	17	83%	85	15	85%
801-900	84	16	84%	84	16	84%
901-1000	75	25	75%	74	26	74%
			82%			82%

TABLE VII. DATASET 6: UNBALANCED STEMMED DATA GENERATED USING R

TEST DATA	STEM		%	NON-STEM		%
	T	***F***		***T***	***F***	
1-100	86	14	86%	86	14	86%
101-200	78	22	78%	77	23	77%
201-300	81	19	81%	84	16	84%
301-400	89	11	89%	86	14	86%
401-500	83	17	83%	83	17	83%
501-600	81	19	81%	77	23	77%
601-700	79	21	79%	79	21	79%
701-800	83	17	83%	85	15	85%
801-900	84	16	84%	84	16	84%
901-1000	75	25	75%	74	26	74%
			82%			82%

TABLE VIII. DATASET 7: BALANCED NON STEMMED DATA GENERATED USING R

TEST DATA	STEM		%	NON-STEM		%
	T	***F***		***T***	***F***	
1-100	83	17	83%	86	14	86%
101-200	77	23	77%	77	23	77%
201-300	82	18	82%	79	21	79%
301-400	87	13	87%	82	18	82%
401-500	82	18	82%	77	23	77%
501-600	77	23	77%	71	29	71%
601-700	77	23	77%	76	24	76%
701-800	83	17	83%	80	20	80%
801-900	83	17	83%	78	22	78%
901-1000	69	31	69%	66	34	66%
			80%			77%

TABLE IX. DATASET 8: BALANCED STEMMED DATA GENERATED USING R

TEST DATA	STEM		%	NON-STEM		%
	T	***F***		***T***	***F***	
1-100	84	16	84%	82	18	82%
101-200	80	20	80%	73	27	73%
201-300	84	16	84%	79	21	79%
301-400	84	16	84%	81	19	81%
401-500	80	20	80%	75	25	75%
501-600	83	17	83%	71	29	71%
601-700	86	14	86%	80	20	80%
701-800	83	17	83%	73	27	73%
801-900	83	17	83%	74	26	74%
901-1000	75	25	75%	65	35	65%
			82%			75%

B. Web Service Comprehensive Testing

a) Method Testing

This test is used to ensure the output of all the web service are according to what we expected in terms of formatting and the content itself. This is the simplest test but also crucial to be performed so that the system gives the same output as what it is expected to do. The result of the method testing can be seen on Table X. All the methods we developed have produced expected results.

TABLE X. METHOD TESTING RESULTS

No	Method	Expected Results	Actual Results	Remarks
1	Version (GET)	JSON 200 OK	JSON 200 OK	Match
2	No (GET)	JSON 200 OK	JSON 200 OK	Match
3	Dataset (GET)	Text Plain 200 OK	Text Plain 200 OK	Match
4	File (GET)	JSON 200 OK	JSON 200 OK	Match
5	Classify (POST)	JSON 200 OK	JSON 200 OK	Match

b) Load Testing

This test is used to see how the system behaves under high load. The instance used in this work is t2 micro which only have 1 vCPU and 1 GB of RAM. In the first test, we used the Version method to represents GET method, with the following parameters:

- Number of threads: 10
- Intervals: 10 s
- Variance: 0.5
- Time limit: 1 s
- Burst delay: 60 s
- Burst duration: 10 s

In load testing, there are 4 sub tests: simple, burst, thread, and variance. The result of the load testing are shown in Fig. 3, Fig. 4, Fig. 5, and Fig. 6.

Test Step	min	max	avg	last	cnt	tps	bytes	bps	err	rat
1 - Request Version	252	3280	521,58	519	229	3,8	20381	338	133	58
TestCase:	252	3280	521,58	519	229	3,8	20381	338	133	58

Fig. 3. Simple Load Testing Result.

Test Step	min	max	avg	last	cnt	tps	bytes	bps	err	rat
1 - Request Version	582	582	582	582	1	1,59	89	142	0	0
TestCase:	582	582	582	582	1	1,59	89	142	0	0

Fig. 4. Burst Load Testing Result.

In simple load test, it has minimal request of 252 ms, maximum request is 3280 ms, and average request is 521,58 ms.

Threads: 10 Strategy Thread Start Threads 1 End Threads 10

Test Step	min	max	avg	last	cnt	tps	bytes	bps	err	rat
1 - Request Version	253	3277	503,58	498	583	16,39	4272	1459	2	0
TestCase:	253	3277	503,58	498	583	16,39	4272	1459	2	0

Fig. 5. Thread Load Testing Result.

Test Step	min	max	avg	last	cnt	tps	bytes	bps	err	rat
1 - Request Version	251	671	494,93	518	912	10,79	1335	960	7	0
TestCase:	251	671	494,93	518	912	10,79	1335	960	7	0

Fig. 6. Variance Load Testing Result.

In burst load test, it has minimal request of 582 ms, maximum request is 582 ms, and average 582 ms.

In thread load test, it has minimal request 253 ms, maximum request is 3277 ms, and average is 503,58 ms. There are 11 requests that has more than 1000 ms (more than time limit of the system). The average request time is around 3 seconds.

In variance load test, it has minimal request of 251 ms, maximum request is 671 ms, and average is 494,93 ms. There are 9 requests that has more than 1000 ms (more than time limit of the system). The average request time is around 3 seconds.

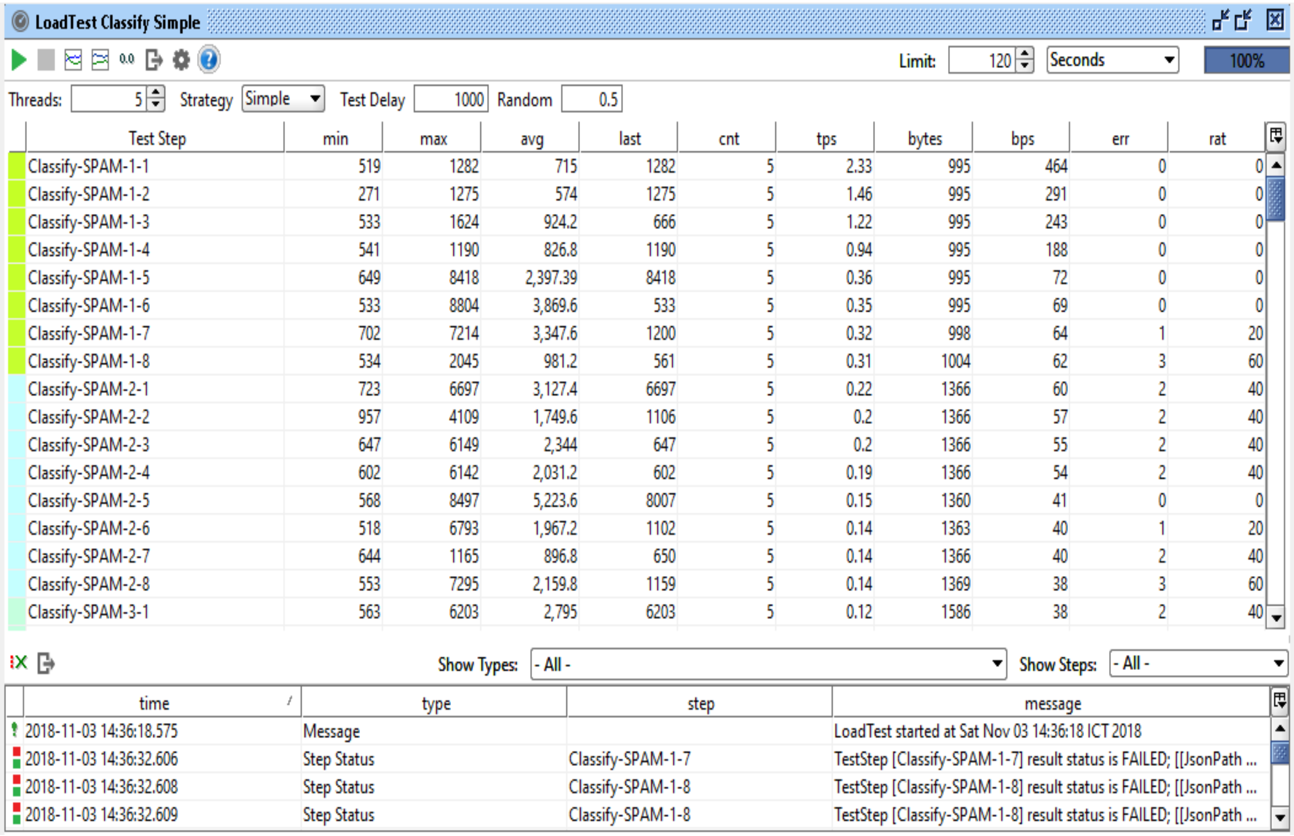
The second test is the Classify method to represents POST method, with the following parameters:

- Number of threads: 5
- Intervals: 60ms
- Variance: 0.5
- Limit: 120-180s
- Burst delay: 10s
- Burst duration: 10s

The result of the load testing is shown in Fig. 7, Fig. 8, Fig. 9, and Fig. 10.

In simple load test, it has minimal request about 0.5-6s, maximum request is 0.5-8ms, and average 0.5-6ms. In thread load test are, it has minimal request about 0.2-4s, maximum request is 0.2-3 ms, and average 0-0.7 ms. The system cannot continue all of the test data as it only finish 19 of 160 test data in 120s. In thread load test, it has minimal request about 0.4-11s, maximum request is 0.4-12ms, and average 0.4-12ms. The system cannot continue to load all the test data as it only finishes 87 of 160 test data in 120s. In variance load test, it has minimal request about 0.4-11s, maximum request is 0.4-12ms, and average 0.4-12ms. The system cannot continue all the test data as it only finished 87 of 160 test data in 120s.

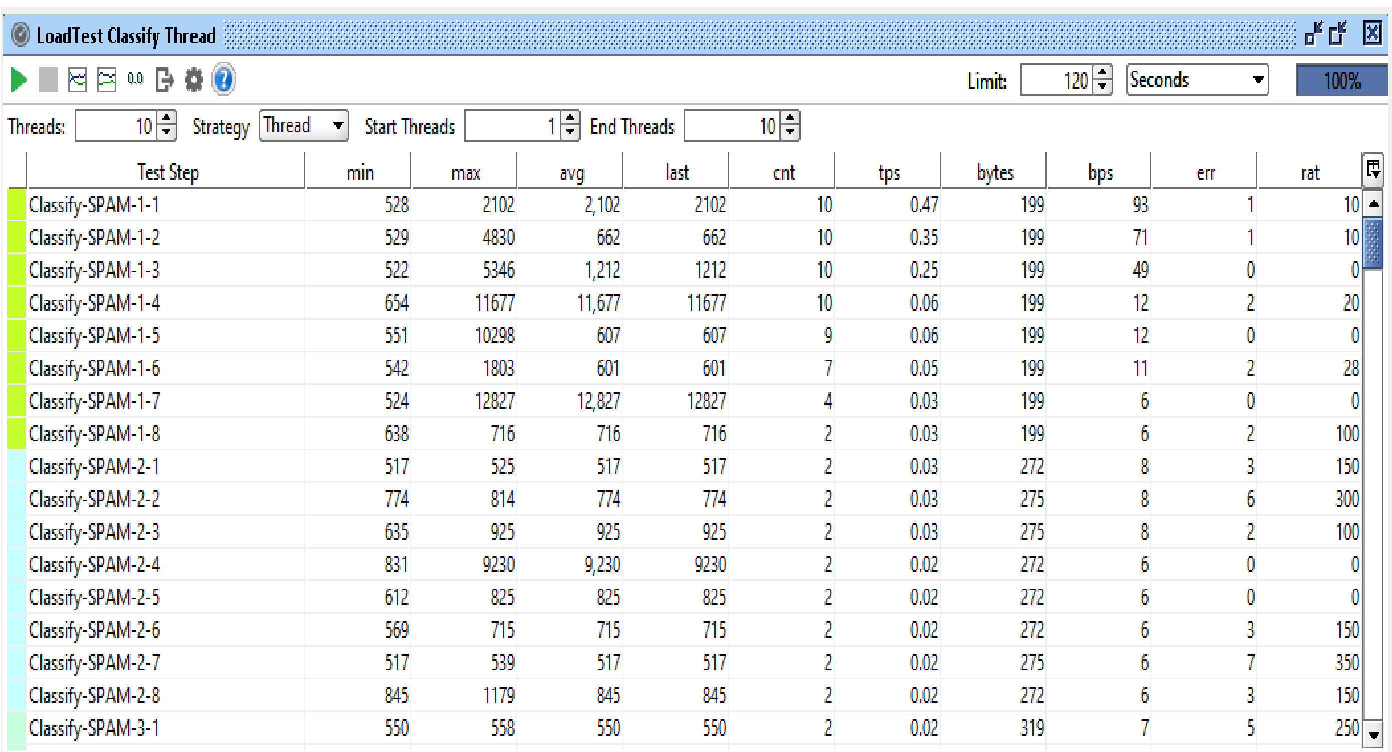
The load testing results are summarized in Table XI and Table XII. The description for the Table are: I is minimum load time, X is maximum load time, and A is Average load time.



Test Step	min	max	avg	last	cnt	tps	bytes	bps	err	rat
Classify-SPAM-1-1	519	1282	715	1282	5	2.33	995	464	0	0
Classify-SPAM-1-2	271	1275	574	1275	5	1.46	995	291	0	0
Classify-SPAM-1-3	533	1624	924.2	666	5	1.22	995	243	0	0
Classify-SPAM-1-4	541	1190	826.8	1190	5	0.94	995	188	0	0
Classify-SPAM-1-5	649	8418	2,397.39	8418	5	0.36	995	72	0	0
Classify-SPAM-1-6	533	8804	3,869.6	533	5	0.35	995	69	0	0
Classify-SPAM-1-7	702	7214	3,347.6	1200	5	0.32	998	64	1	20
Classify-SPAM-1-8	534	2045	981.2	561	5	0.31	1004	62	3	60
Classify-SPAM-2-1	723	6697	3,127.4	6697	5	0.22	1366	60	2	40
Classify-SPAM-2-2	957	4109	1,749.6	1106	5	0.2	1366	57	2	40
Classify-SPAM-2-3	647	6149	2,344	647	5	0.2	1366	55	2	40
Classify-SPAM-2-4	602	6142	2,031.2	602	5	0.19	1366	54	2	40
Classify-SPAM-2-5	568	8497	5,223.6	8007	5	0.15	1360	41	0	0
Classify-SPAM-2-6	518	6793	1,967.2	1102	5	0.14	1363	40	1	20
Classify-SPAM-2-7	644	1165	896.8	650	5	0.14	1366	40	2	40
Classify-SPAM-2-8	553	7295	2,159.8	1159	5	0.14	1369	38	3	60
Classify-SPAM-3-1	563	6203	2,795	6203	5	0.12	1586	38	2	40

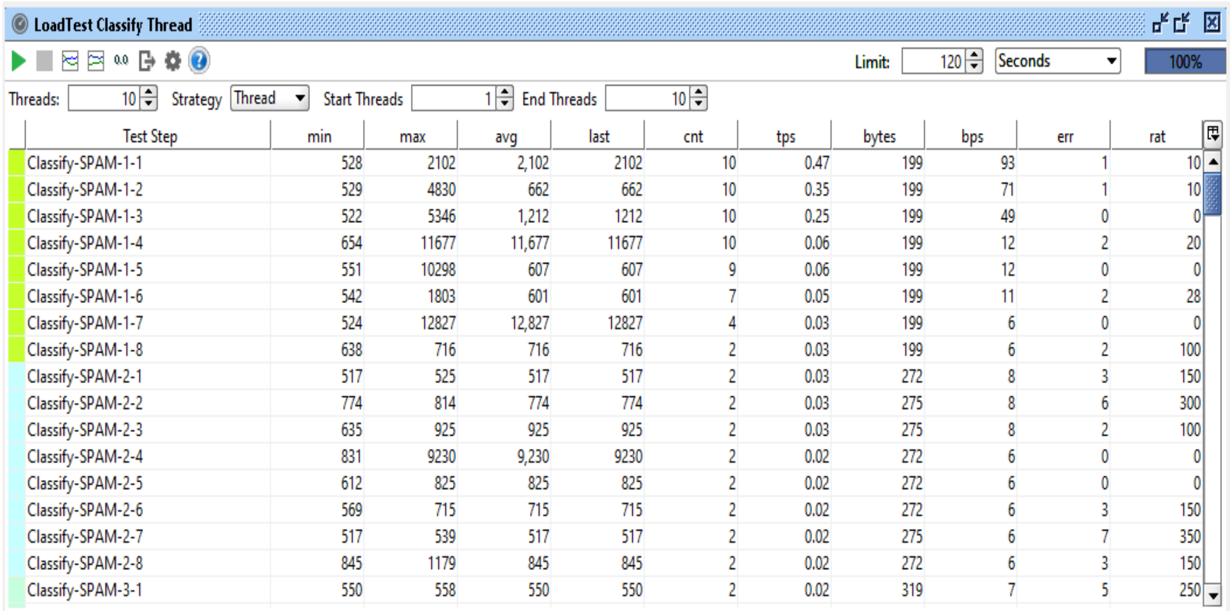
time	type	step	message
2018-11-03 14:36:18.575	Message		LoadTest started at Sat Nov 03 14:36:18 ICT 2018
2018-11-03 14:36:32.606	Step Status	Classify-SPAM-1-7	TestStep [Classify-SPAM-1-7] result status is FAILED; [[JsonPath ...
2018-11-03 14:36:32.608	Step Status	Classify-SPAM-1-8	TestStep [Classify-SPAM-1-8] result status is FAILED; [[JsonPath ...
2018-11-03 14:36:32.609	Step Status	Classify-SPAM-1-8	TestStep [Classify-SPAM-1-8] result status is FAILED; [[JsonPath ...

Fig. 7. Simple Load Testing on Classify Method.



Test Step	min	max	avg	last	cnt	tps	bytes	bps	err	rat
Classify-SPAM-1-1	528	2102	2,102	2102	10	0.47	199	93	1	10
Classify-SPAM-1-2	529	4830	662	662	10	0.35	199	71	1	10
Classify-SPAM-1-3	522	5346	1,212	1212	10	0.25	199	49	0	0
Classify-SPAM-1-4	654	11677	11,677	11677	10	0.06	199	12	2	20
Classify-SPAM-1-5	551	10298	607	607	9	0.06	199	12	0	0
Classify-SPAM-1-6	542	1803	601	601	7	0.05	199	11	2	28
Classify-SPAM-1-7	524	12827	12,827	12827	4	0.03	199	6	0	0
Classify-SPAM-1-8	638	716	716	716	2	0.03	199	6	2	100
Classify-SPAM-2-1	517	525	517	517	2	0.03	272	8	3	150
Classify-SPAM-2-2	774	814	774	774	2	0.03	275	8	6	300
Classify-SPAM-2-3	635	925	925	925	2	0.03	275	8	2	100
Classify-SPAM-2-4	831	9230	9,230	9230	2	0.02	272	6	0	0
Classify-SPAM-2-5	612	825	825	825	2	0.02	272	6	0	0
Classify-SPAM-2-6	569	715	715	715	2	0.02	272	6	3	150
Classify-SPAM-2-7	517	539	517	517	2	0.02	275	6	7	350
Classify-SPAM-2-8	845	1179	845	845	2	0.02	272	6	3	150
Classify-SPAM-3-1	550	558	550	550	2	0.02	319	7	5	250

Fig. 8. Burst Load Testing of Classify Method.

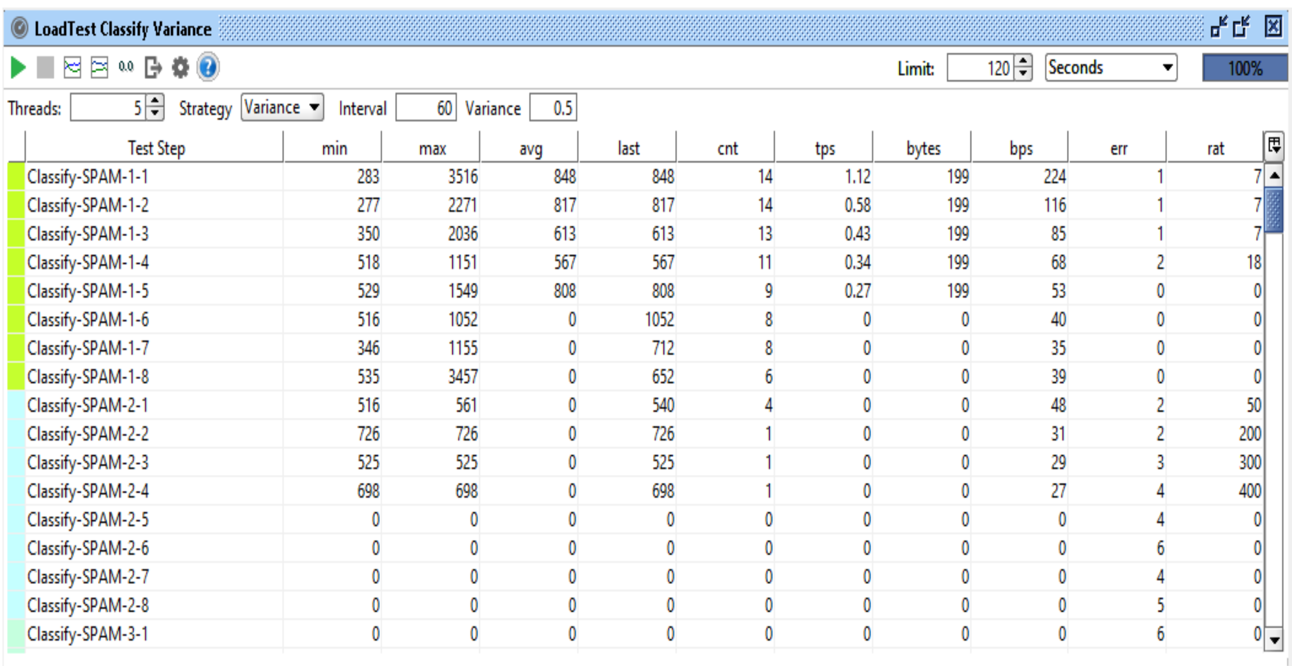


LoadTest Classify Thread

Threads: 10 Strategy Thread Start Threads 1 End Threads 10 Limit: 120 Seconds 100%

Test Step	min	max	avg	last	cnt	tps	bytes	bps	err	rat
Classify-SPAM-1-1	528	2102	2,102	2102	10	0.47	199	93	1	10
Classify-SPAM-1-2	529	4830	662	662	10	0.35	199	71	1	10
Classify-SPAM-1-3	522	5346	1,212	1212	10	0.25	199	49	0	0
Classify-SPAM-1-4	654	11677	11,677	11677	10	0.06	199	12	2	20
Classify-SPAM-1-5	551	10298	607	607	9	0.06	199	12	0	0
Classify-SPAM-1-6	542	1803	601	601	7	0.05	199	11	2	28
Classify-SPAM-1-7	524	12827	12,827	12827	4	0.03	199	6	0	0
Classify-SPAM-1-8	638	716	716	716	2	0.03	199	6	2	100
Classify-SPAM-2-1	517	525	517	517	2	0.03	272	8	3	150
Classify-SPAM-2-2	774	814	774	774	2	0.03	275	8	6	300
Classify-SPAM-2-3	635	925	925	925	2	0.03	275	8	2	100
Classify-SPAM-2-4	831	9230	9,230	9230	2	0.02	272	6	0	0
Classify-SPAM-2-5	612	825	825	825	2	0.02	272	6	0	0
Classify-SPAM-2-6	569	715	715	715	2	0.02	272	6	3	150
Classify-SPAM-2-7	517	539	517	517	2	0.02	275	6	7	350
Classify-SPAM-2-8	845	1179	845	845	2	0.02	272	6	3	150
Classify-SPAM-3-1	550	558	550	550	2	0.02	319	7	5	250

Fig. 9. Thread Load Testing of Classify Method.



LoadTest Classify Variance

Threads: 5 Strategy Variance Interval 60 Variance 0.5 Limit: 120 Seconds 100%

Test Step	min	max	avg	last	cnt	tps	bytes	bps	err	rat
Classify-SPAM-1-1	283	3516	848	848	14	1.12	199	224	1	7
Classify-SPAM-1-2	277	2271	817	817	14	0.58	199	116	1	7
Classify-SPAM-1-3	350	2036	613	613	13	0.43	199	85	1	7
Classify-SPAM-1-4	518	1151	567	567	11	0.34	199	68	2	18
Classify-SPAM-1-5	529	1549	808	808	9	0.27	199	53	0	0
Classify-SPAM-1-6	516	1052	0	1052	8	0	0	40	0	0
Classify-SPAM-1-7	346	1155	0	712	8	0	0	35	0	0
Classify-SPAM-1-8	535	3457	0	652	6	0	0	39	0	0
Classify-SPAM-2-1	516	561	0	540	4	0	0	48	2	50
Classify-SPAM-2-2	726	726	0	726	1	0	0	31	2	200
Classify-SPAM-2-3	525	525	0	525	1	0	0	29	3	300
Classify-SPAM-2-4	698	698	0	698	1	0	0	27	4	400
Classify-SPAM-2-5	0	0	0	0	0	0	0	0	4	0
Classify-SPAM-2-6	0	0	0	0	0	0	0	0	6	0
Classify-SPAM-2-7	0	0	0	0	0	0	0	0	4	0
Classify-SPAM-2-8	0	0	0	0	0	0	0	0	5	0
Classify-SPAM-3-1	0	0	0	0	0	0	0	0	6	0

Fig. 10. Variance Load Testing of Classify Method.

TABLE XI. SUMMARY OF SIMPLE AND BURST TESTING

Methods	Average (in milliseconds)					
	Simple			***Burst***		
	I	***X***	***A***	***I***	***X***	***A***
GET Version	252	3280	521	582	582	582
POST Classify	1255	7403	3355	404	1425	386

TABLE XII. SUMMARY OF THREAD AND VARIANCE TESTING

Methods	Average (in milliseconds)					
	Thread			***Variance***		
	I	***X***	***A***	***I***	***X***	***A***
GET Version	253	3277	503	251	671	494
POST Classify	913	1547	1319	484	1558	304

Version method is considerably faster than Classify method because it only returns static text, while Classify method is more slower because it does spam detection process. This characteristics is also shown in the simple and burst testing and thread and variance testing results. The Classify method performance is also affected by the length of the input and the size of the datasets used for spam detection process.

c) Data Driven Testing

Data driven test is using test data that has been stored in some external storage and use it iteratively. 8 datasets were used in which each dataset consists of 20 test data and divided into 2 more categories: 10 data categorized as SPAM and 10 data categorized as NON-SPAM so in total, it has 160 tests. The metrics measured were response time and accuracy. The results can be seen in Table XIII.

The result of this accuracy on data driven test with SOAPUI are: the accuracy is 63.125 % and average response time is about 2 seconds.

C. Browser Extension Development

The browser extension was developed extensively for Mozilla Firefox since it was using Greasemonkey plugin although it is also working in Google Chrome.

The extension is dynamically detecting the URL loaded in the address bar. If it is coming from Instagram's URL, it will add a new entry in the context menu (right click menu) as the user highlight some comment as shown in Fig. 11. When user clicked the entry, it will send the text to the Classify method in our web services and it will return the results ('spam' or 'not spam') in clear text and show it to user Fig. 12. In Google Chrome, the results are displayed as inFig. 13.

The browser extension developed is working as expected and able to do the spam detection process utilizing REST-based web service that were deployed in earlier work. The extension's user interface still need some improvements to make it easier to use for common user.

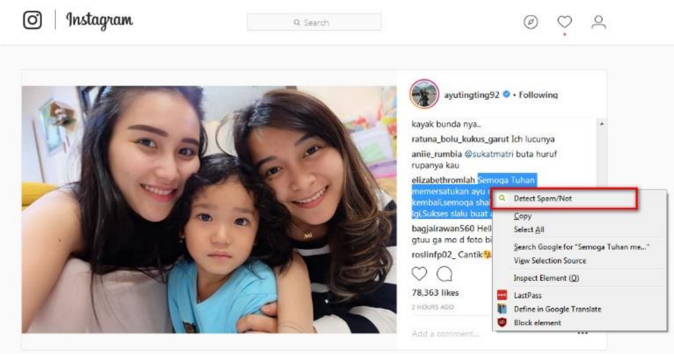


Fig. 11. New Entry in Firefox's Context Menu.

TABLE XIII. DATA DRIVEN TESTING RESULT

STEP	TEST ID	RESULT	CATEGORY	TIME
Step 1	[Classify-SPAM-1-1]	OK	SPAM	1162 ms
Step 2	[Classify-SPAM-1-2]	OK	SPAM	521 ms
Step 3	[Classify-SPAM-1-3]	OK	SPAM	1638 ms
Step 4	[Classify-SPAM-1-4]	FAILED	NONSPAM	1543 ms
Step 5	[Classify-SPAM-1-5]	OK	SPAM	2208 ms
Step 6	[Classify-SPAM-1-6]	OK	SPAM	2271 ms
Step 7	[Classify-SPAM-1-7]	OK	SPAM	1554 ms
Step 8	[Classify-SPAM-1-8]	OK	SPAM	1921 ms
Step 9	[Classify-SPAM-2-1]	OK	SPAM	2468 ms
Step 10	[Classify-SPAM-2-2]	FAILED	NONSPAM	1125 ms
...	...	...	...	...
...	...	...	...	...
Step 150	[Classify-NOSPAM-9-6]	FAILED	SPAM	2211 ms
Step 151	[Classify-NOSPAM-9-7]	FAILED	SPAM	1512 ms
Step 152	[Classify-NOSPAM-9-8]	FAILED	SPAM	1883 ms
Step 153	[Classify-NOSPAM-10-1]	FAILED	SPAM	1964 ms
Step 154	[Classify-NOSPAM-10-2]	OK	NONSPAM	1123 ms
Step 155	[Classify-NOSPAM-10-3]	OK	NONSPAM	4039 ms
Step 156	[Classify-NOSPAM-10-4]	OK	NONSPAM	1660 ms
Step 157	[Classify-NOSPAM-10-5]	OK	NONSPAM	1988 ms
Step 158	[Classify-NOSPAM-10-6]	FAILED	SPAM	3511 ms
Step 159	[Classify-NOSPAM-10-7]	FAILED	SPAM	1863 ms
Step 160	[Classify-NOSPAM-10-8]	FAILED	SPAM	1797 ms
AVERAGE ACCURACY / TIME			**63.125%**	**1991,244 ms**

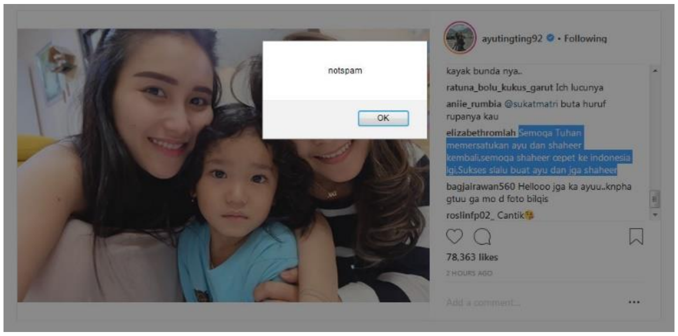


Fig. 12. Result of Classify Method in Mozilla Firefox.

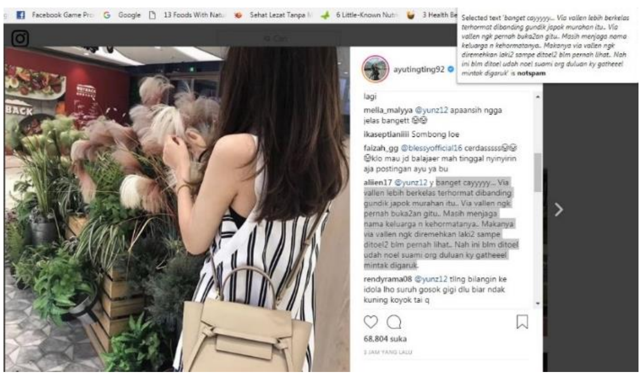


Fig. 13. Result of Classify Method in Google Chrome.

V. Conclusions

In this paper, a browser extension for Firefox & Chrome has been successfully developed and integrated into a REST-based web service [19] deployed on top of AWS Platform. Accuracy of the web service were measured using three datasets (whole datasets, 1000 stemmed dataset and 1000 non-stemmed dataset) and achieved accuracy level of 63.125% for whole datasets, 72% for non-stemmed dataset, and 70% for stemmed dataset. The average response time is under 2s, minimum load time test is between 0.2 – 1.2s, and, maximum load time test is between 3 – 7s. Although the browser extension is working as expected, the user interface and data accuracy still have room for improvements.

Acknowledgment

This research is partially funded by UKDW Research and Community Service (LPPM) unit under grant no. 072/D.01/LPPM/2018 and Faculty of Information Technology.

References

[1] L. Vries, S. Gensler and P. S. Leeflang, "Popularity of Brand Posts on Brand Fan Pages: An Investigation of the Effects of Social Media Marketing," Journal of Interactive Marketing, vol. 26, no. 2, pp. 83-91, 2012.

[2] Z. Xiang and U. Gretzel, "Role of social media in online travel information search," Tourism Management, vol. 31, no. 2, pp. 179-188, 2010.

[3] W.-y. S. Chou, Y. M. Hunt, E. B. Beckjord, R. P. Moser and B. W. Hesse, "Social Media Use in the United States: Implications for Health Communication," Journal of Medical Internet Research, vol. 11, no. 4, 2009.

[4] M. Chakraborty, S. Pal, R. Pramanik and C. R. Chowdary, "Recent developments in social spam detection and combating techniques: A survey," Information Processing & Management, vol. 52, no. 6, pp. 1053-1073, 2016.

[5] M. Salehi, S. Shehnepoor, R. Farahbakhsh and N. Crespi, "NetSpam: A Network-Based Spam Detection Framework for Reviews in Online Social Media.," in IEEE Transactions on Information Forensics and Security, 2017.

[6] W. Zhang and H. M. Sun, "Instagram spam detection," in 22nd IEEE Pacific Rim International Symposium on Dependable Computing, Christchurch, New Zealand, 2017.

[7] A. R. Chrismanto and Y. Lukito, "Klasifikasi Komentar Spam Pada Instagram Berbahasa Indonesia," in Seminar Nasional Teknologi Informasi Kesehatan (SNATIK), Yogyakarta, 2017.

[8] F. Fathaliani and M. Bouguessa, "A Model-Based Approach for Identifying spammers in social networks," in IEEE International Conference on Data Science and Advanced Analytics (DSAA), Paris, France, 2015.

[9] N. Agarwal and Y. Yiliyasi, "Information quality challenges in social media," in The 15th International Conference on Information Quality, Little Rock, Arkansas, USA, 2010.

[10] J. Constine, "TechCrunch," 20 June 2018. [Online]. Available: https://techcrunch.com/2018/06/20/instagram-1-billion-users/. [Accessed 1 November 2018].

[11] R. Hardinata and J. Tirtawangsa, "A game with purpose to filter spams from Indonesian Twitter trending topics," in 2016 4th International Conference on Information and Communication Technology (ICoICT), Bandung, Indonesia, 2016.

[12] A. A. Septiandri and O. Wibisono, "Detecting spam comments on Indonesia's Instagram posts," Journal of Physics: Conference Series, vol. 801, no. 1, 2017.

[13] A. Barth, "The Web Origin Concept," Infosec Institute, December 2001. [Online]. Available: https://tools.ietf.org/html/rfc6454. [Accessed 1 November 2018].

[14] W. S. Raharjo and A. Ashari, "IMPLEMENTASI ANNOTEA CLIENT BERBASIS WEB UNTUK MENGATASI ATURAN SAME ORIGIN POLICY," in KNASTIK, Yogyakarta, Indonesia, 2009.

[15] Z. Wuxain and S. Hung-Min, "Instagram Spam Detection," in IEEE 22nd Pacific Rim International Symposium on Dependable Computing (PRDC), Christchurch, New Zealand, 2017.

[16] A. Rachmat and Y. Lukito, "Deteksi Komentar Spam Bahasa Indonesia Pada Instagram Menggunakan Naive Bayes," Ultimatics, vol. 9, no. 1, 2017.

[17] A. R. Chrismanto and Y. Lukito, "KLASIFIKASI KOMENTAR SPAM PADA INSTAGRAM BERBAHASA INDONESIA MENGGUNAKAN K-NN," in Seminar Nasional Teknologi Informasi Kesehatan (SNATIK) 2017, Yogyakarta, 2017.

[18] A. R. Chrismanto and Y. Lukito, "Identifikasi Komentar Spam Pada Instagram," Lontar Komputer, vol. 8, no. 3, pp. 219-231, 2017.

[19] A. R. Chrismanto, W. S. Raharjo and Y. Lukito, "Design and Development of REST-based Instagram Spam Detector for Indonesian Language," in 3rd International Seminar on Application for Technology of Information and Communication (iSemantic), Semarang, Indonesia, 2018.

[20] R. Fielding, Architectural Styles and the Design of Network-based Software, California: University of California, 2000.

[21] S. Malik and D.-H. Kim, "A comparison of RESTful vs. SOAP web services in actuator networks," in 2017 Ninth International Conference on Ubiquitous and Future Networks (ICUFN), Milan, Italy, 2017.

[22] S. Kumari and S. K. Rath, "Performance comparison of SOAP and REST based Web Services for Enterprise Application Integration," in 2015 International Conference on Advances in Computing, Communications and Informatics (ICACCI), Kochi, India, 2015.