AGREGASI TOKO APLIKASI PADA PIRANTI BERGERAK UNTUK MEMUDAHKAN PENCARIAN APLIKASI MULTI PLATFORM

Rully Agus Hendrawan[1)], Andre Parvian Aristio, dan Pri Rezki Destrianto
Departemen Sistem Informasi, Fakultas Teknologi Informasi dan Komunikasi,
Institut Teknologi Sepuluh Nopember (ITS)
Jl. Jalan Raya ITS, Surabaya 60111, Indonesia
Telp : (031) 5999944, Fax : (031) 596 4965
E-mail: eraha@is.its.ac.id[1)]

Abstrak

Mobile application users need easier way to understand and evaluate applications according to their individual needs and mobile devices they owned. Getting necessary information before deciding whether users want to install the application, require extra effort because the information are widespread on application store, blogs, websites, and social media. That information is not well structured and not consistent. Application directory helps provide integrated information about application taken from several application stores. Thus, helps users to search and get comprehensif information. For application developers, directory could also help publishing and promoting their application. Method that being used in this research is prototyping. We develop a prototype that aggregate detailed information about mobile applications from two biggest application stores. The output of this research is web based application that provide structured catalog of multi-platform mobile application.

Para pengguna aplikasi (perangkat lunak) menginginkan cara yang lebih mudah dan cepat dalam melakukan evaluasi berbagai aplikasi yang sesuai dengan kebutuhannya dan gawai yang dimiliki. Pengguna aplikasi mengalami kesulitan memilih karena informasi yang tersebar di toko, blog, website, dan media sosial. Informasi tersebut kurang terstruktur dan tidak konsisten. Direktori aplikasi membantu menyajikan daftar aplikasi dari beberapa toko aplikasi secara terintegrasi sehingga memudahkan pengguna dalam melakukan pencarian aplikasi dan memperoleh informasi yang komprehensif. Bagi pengembang aplikasi, direktori juga dapat menjadi media publikasi karya aplikasi, ajang eksistensi, penghargaan, dan promosi. Metode yang digunakan dalam penelitian ini adalah prototyping dengan melakukan agregasi detail informasi aplikasi dari dua toko aplikasi mobile yang berbeda platform ke dalam satu wadah katalog. Luaran penelitian ini berupa sebuah prototipe aplikasi berbasis web yang menyajikan katalog aplikasi piranti bergerak yang terstruktur. Hasil yang diharapkan dalam penelitian ini adalah untuk membantu pengguna dalam mencari aplikasi multiplatform sesuai dengan kebutuhan.

***Kata Kunci**: agregasi produk, direktori perangkat lunak, software development*

1. PENDAHULUAN

Meskipun aplikasi untuk piranti bergerak telah ada semenjak tahun 90an, tren tersebut baru mulai meningkat ketika Apple mempublikasikan toko aplikasinya yang terintegrasi dalam gawai produksinya. Tren ini terus meningkat hingga tahun 2014, dimana pengguna aplikasi piranti bergerak telah melebihi pengguna aplikasi desktop [1]. Toko aplikasi pada dasarnya adalah platform distribusi daring dimana pengguna dapat mengunduh dan memasang aplikasi yang dikembangkan oleh pihak ketiga untuk gawai dan sistem operasi yang didukung platform tersebut. Saat ini, terdapat dua toko aplikasi yang mendominasi 90% pengunduhan aplikasi secara global yaitu App Store milik Apple dan Play Store milik Google [2].

Seiring bertambahnya jumlah aplikasi yang melimpah pada layanan konten digital, proses pencarian aplikasi yang spesifik akan semakin sulit. Para pengguna aplikasi membutuhkan cara yang efektif dan efisien dalam melakukan evaluasi berbagai aplikasi yang tersedia dengan cepat dan sesuai dengan kebutuhannya masing-masing. Misalnya, ketika pengguna sangat menyukai sebuah aplikasi tertentu dan ingin tahu pada platform apa saja aplikasi tersebut tersedia. Informasi ini juga akan membantu pengguna

ketika memilih gawai, dengan mengetahui sebelumnya seberapa bagus dukungan aplikasi terhadap platform yang ada pada gawai tersebut.

Keberadaan platform turut berperan dalam kesuksesan aplikasi, misalnya penentuan harga [3]. Toko aplikasi menjadi sumber utama dalam menemukan aplikasi baru yaitu dengan cara mencari (20%) dan rekomendasi sistem (13%). Namun, sering kali pemilik gawai memilih mencari informasi dari tempat lain seperti rekomendasi teman (15%), media sosial (10%), berita (8%), website (10%), iklan daring (9%) , dan iklan luring (6%) [4]. Hal ini membuat para pengembang aplikasi harus menyisihkan dana untuk membangun website promosi. Keberadaan direktori aplikasi dapat turut membantu mempublikasikan karya pengembang aplikasi. Agregasi informasi aplikasi juga dilakukan peneliti lain untuk komparasi aplikasi berdasar ulasan [5], sistem rekomendasi [6], menghasilkan model pengetahuan [7], opinion mining [8].

Untuk menangani permasalahan tersebut, penelitian ini berfokus pada pengembangan sebuah prototipe direktori aplikasi yang melakukan penyajian katalog aplikasi kepada pengguna dengan melakukan agregasi produk dari dua toko yaitu seperti Google Play Store dan Apple App Store. Metode yang dilakukan dalam penelitian ini adalah prototyping. Prototyping adalah metodologi pengembangan perangkat lunak di mana model sistem dikembangkan dan dievaluasi sesering mungkin. Semakin banyak pengujian membantu menentukan kebutuhan sistem dan kemampuan fungsional dengan lebih cepat dan akurat. Metode prototyping digunakan karena fleksibilitas dalam pembangunan dari desain aplikasi yang akan dibangun.

Dengan adanya prototipe ini diharapkan dapat mempermudah pengguna gawai dalam mendapatkan informasi aplikasi multiplatform secara detail dan lengkap. Selain itu, dapat menjadi bahan pertimbangan dalam pembelian gawai di masa yang akan datang. Bagi developer, prototipe ini dapat membantu publikasi dan pemasaran produknya.

2. TINJAUAN PUSTAKA

Bab ini akan menjelaskan mengenai dasar teori yang dijadikan acuan atau landasan dalam pengerjaan penelitian ini. Landasan teori akan memberikan gambaran umum dari landasan berpikir penelitian ini.

2.1 Levenshtein Distance

Levenshtein Distance (LD) merupakan sebuah ukuran dari kemiripan atau kecocokan antar dua teks. Misalnya, terdapat teks acuan sebagai sumber atau disebut s dan target atau disebut t [9]. Nilai jarak diperoleh dengan menghitung jumlah langkah kumulatif dari tindakan penghapusan (deletion), penambahan (insertion) atau penggantian (substitution) yang dibutuhkan untuk merubah s menjadi t. Contohnya adalah sebagai berikut:

- Jika s adalah ”test” dan t adalah ”test”, maka LD(s,t) = 0, karena tidak ada transformasi yang dibutuhkan dimana s = t.
- Jika s adalah ”test” dan t adalah ”tent”, maka LD(s,t) = 1, karena dibutuhkan 1 substitusi untuk membuat s = t, sehingga nilai dari Levenshtein adalah 1.

Secara umum, algoritma dari Levenshtein Distance berdasarkan dijelaskan pada Tabel 1.

Table 1. Pseudocode Levenshtein Distance

	Deskripsi
1	*n* = panjang dari *s* *m* = panjang dari *t* Jika *n* = 0, kembalikan nilai *m* dan keluar. Jika *m = 0*, kembalikan nilai *n* dan keluar. Bangun matriks berisikan baris 0..*m*, dan kolom 0..n
2	Inisialisasi baris pertama dengan 0..n. Inisialisasi kolom pertama dengan 0..m.
3	Cek tiap huruf pada *s* (dengan *i* dari 1 sampai dengan *n*).
4	Cek tiap huruf pada *t* (dengan *j* dari 1 sampai dengan *m*).
5	Jika huruf *s*[*i*] sama dengan *t*[*j*], maka *biaya* = 0. Jika huruf *s*[*i*] tidak sama dengan *t*[*j*], maka *biaya* = 1.
6	Tetapkan nilai *d*[*i,j*] dari matriks dengan nilai minimum dari: a. Nilai elemen yang berada tepat diatasnya ditambah satu: *d*[*i*-1, *j*] + 1. b. Nilai elemen yang berada tepat dikirinya ditambah satu: *d*[*i*, *j*-1] + 1. c. Nilai elemen yang berada pada diagonal kiri atas ditambah biaya: *d*[*i*-1, *j*-1] + *biaya*

7	Setelah semua langkah iterasi (3, 4, 5, 6) berakhir, jarak dapat ditemukan pada nilai d[n,m]

2.2 Web Scraping

Cara yang umum digunakan dalam mengambil data dari laman web adalah dengan menyalin/tempel sebuah menggunakan peramban. *Web scraping* adalah mengambil konten *website* secara otomatis tanpa harus berinteraksi dengan laman tersebut menggunakan peramban, untuk kemudian dipilah-pilah sesuai dengan informasi yang dibutuhkan [10]. Pada penelitian ini, tujuan dari *web scraping* adalah untuk mengumpulkan data terkait aplikasi pada *App Store (Apple)* dan juga *Play Store (Google)* dan kemudian menyimpannya kedalam format yang terstruktur.

3. METODOLOGI

Metodologi merupakan sebuah tahapan dalam penyelesaian permasalahan pada penelitian ini. Metodologi digunakan sebagai panduan pengerjaan penelitian.

Gambar 1. Metodologi Penelitian

3.1 Analisis Kebutuhan

Analisis kebutuhan dilakukan dengan menyebarkan kuesioner kepada 30 orang yang berpengalaman dalam memilih dan memasang perangkat lunak pada gawai. Mereka sangat memahami cara mengoperasikan *App Store (Apple)* dan juga *Play Store (Google).* Mereka juga cukup rutin memasang perangkat lunak baru pada gawai miliknya. Selain itu, dilakukan juga wawancara kepada 2 orang pengembang perangkat lunak piranti bergerak untuk menggali kebutuhannya terkait metode publikasi dan promosi karyanya. Hasil wawancara nantinya akan digunakan sebagai bahan untuk menentukan kebutuhan fungsional sistem yang akan dirancang agar sesuai dengan kebutuhan pengguna.

3.2 Pengambilan Data

Setelah melakukan analisis kebutuhan, lalu dilakukan pengambilan data yang dilakukan dengan *web scraping*. Data yang diambil berasal dari *App Store (Apple)* dan juga *Play Store (Google).* Data yang didapatkan meliputi nama aplikasi dan detail deskripsi aplikasi yang ada dalam layanan konten digital. Pengumpulan data dilakukan dengan memanfaatkan *Application Programming Interface (API)* dari layanan *middleware* yang melakukan *web scraping*. Data yang telah dikumpulkan ditransformasi ke dalam sistem basis data relasional [11].

3.3 Pengujian Prototipe

Setelah menyelesaikan pengambilan data, maka tahap selanjutnya adalah pembuatan prototipe *website* agregasi produk dengan fungsi yang spesifik terhadap direktori aplikasi (perangkat lunak). Tujuan utama dari proses ini adalah untuk memberikan pengalaman terbaik dalam melakukan evaluasi/pemilihan aplikasi piranti bergerak. Prototipe berupa aplikasi web berbasis teknologi Apache, PHP, dan MySql yang menyajikan direktori aplikasi secara terstruktur. Prototipe dibuat dengan merancang draft antar muka pengguna yang merepresentasikan kebutuhan fungsional. Langkah selanjutnya dilakukan perancangan dengan membuat desain perangkat lunak.

3.4 Pengujian Aplikasi

Pada tahapan ini prototipe akan diuji usabilitas, luaran, dan fungsionalnya. Pengujian usabilitas menggunakan SUS *matrix* [12], [13], dilakukan kepada 10 orang responden. Apabila dari hasil pengujian didapatkan nilai dibawah dari ketentuan maka desain dan fitur diubah sesuai dengan harapan pengguna. Selain itu mungkin akan didapatkan pula masukan di luar konteks penelitian, sehingga dapat menjadi bahan pengembangan di masa yang akan datang.

4. HASIL DAN PEMBAHASAN

Pada tahapan ini berisi hasil implementasi penelitian yang telah dilaksanakan dan hasil pengujian prototipe yang dibuat berdasarkan fitur yang ada dalam aplikasi.

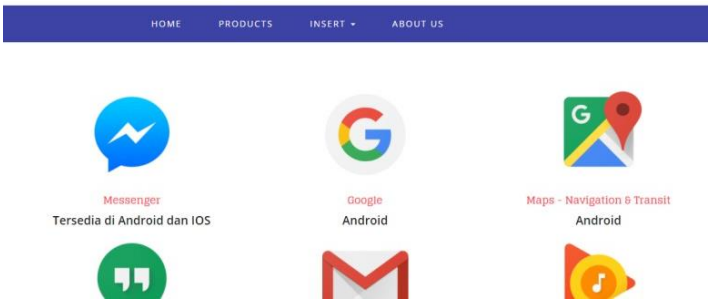
4.1 Hasil Prototipe Aplikasi

Hasil prototipe berupa aplikasi web yang siap dipakai dengan kapabilitas terbatas pada kemampuan

fungsional utama yaitu menampilkan daftar aplikasi yang telah diagregasi, melihat detail produk beserta platform yang didukung, memasukkan data aplikasi yang dibuat oleh developer serta otentikasi dan pendaftaran untuk setiap pengguna non-publik. Terdapat enam tampilan antar muka antara lain halaman katalog aplikasi, halaman detail aplikasi, halaman masukan data aplikasi berdasarkan identitas, masukan data secara rinci, halaman otentikasi, dan halaman registrasi. Selanjutnya akan dijelaskan masing-masing fitur halaman utama.

4.1.1 Halaman Daftar Aplikasi

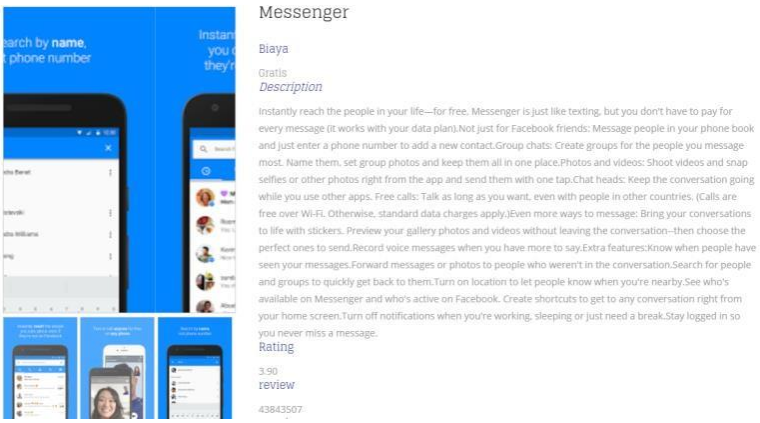
Gambar 2 menunjukkan tampilan awal dari prototipe yang menampilkan daftar aplikasi dari hasil *scraping* yang berhasil diambil. Pengguna bisa mengetahui aplikasi tersebut tersedia di *Apple App Store* maupun *Google Play Store* dari label yang ada dibawah judul. Label ini dihasilkan dari deteksi yang dijelaskan pada bagian 4.2.



Gambar 2. Halaman Daftar Aplikasi

4.1.2 Halaman Detail Aplikasi

Gambar 3 adalah tampilan untuk menampilkan informasi aplikasi secara rinci. Pengguna bisa mengetahui informasi aplikasi secara lengkap dan disediakan *screenshoot* aplikasi. Tersedia juga *hyperlink* yang menuju toko untuk memasang aplikasi tersebut.



Gambar 3. Halaman Detail Aplikasi

4.1.3 Halaman Masukan Data Secara Rinci

Gambar 4 adalah tampilan untuk memasukkan data aplikasi secara rinci. Pengguna yang belum pernah memasukkan karyanya ke toko dapat memasukkan aplikasi yang dibuat ke formulir di halaman ini dengan mengisi data-data yang ada di formulir tersebut. Aplikasi yang sudah masuk dalam toko tentunya akan diambil datanya secara otomatis menggunakan *web scraping*.

Gambar 4. Halaman masukan data secara rinci

4.1.4 Halaman Masukan Data Berdasarkan Appid

Gambar 5 adalah tampilan untuk memasukkan appId aplikasi yang tersedia di toko. Pengguna yang mempunyai karya berupa aplikasi dan sudah diunggah ke *Apple App Store* atau *Google Play Store* dapat memasukkan appId aplikasi dan sistem secara otomatis akan melakukan *scraping*.

Gambar 5. Halaman masukan data berdasarkan appid

4.1.5 Deteksi Aplikasi yang Sama

Proses *web scraping* mengambil data dari *Google Play Store* sejumlah 44.100 aplikasi dan dari *Apple App Store* sejumlah 18.318 aplikasi. Dari kedua toko tersebut, diambil masing-masing 2.500 sampel aplikasi untuk ujicoba agregasi informasi dengan menggabungkan aplikasi yang sama menjadi satu *tuple* (baris). Agregasi dilakukan dengan melakukan deteksi kesamaan teks appId (kode identitas aplikasi), judul aplikasi, dan nama developer. Aplikasi dari kedua toko dengan nilai kesamaan teks yang bagus akan dianggap sebagai aplikasi yang sama. Terdapat tiga relasi dalam basis data yang dipakai dalam proses ini, yaitu:
playstore_apps(appId_app, title_app, developer_app, is_checked) adalah relasi/tabel berisikan aplikasi dari *Google Play Store*; appstore_apps(app_idplay, title_play, developer_play) adalah relasi berisikan aplikasi dari *Apple App Store*; dan aggregate_apps(appId_app, app_idplay, title_app, title_play, status) adalah relasi berisikan hasil penggabungan kedua relasi playstore_apps dan appstore_apps.
Penjelasan masing-masing atribut dari relasi ditujukkan pada Tabel 3.

Table 3. Atribut Relasi untuk Deteksi Aplikasi

	Relasi	Atribut	Deskripsi
1	appstore_apps	appId_app	identitas aplikasi pada *Apple App Store*
2	appstore_apps	title_app	judul aplikasi pada *Apple App Store*
3	appstore_apps	developer_app	nama perusahaan developer aplikasi pada *Apple App Store*
4	playstore_apps	app_idplay	identitas aplikasi pada *Google Play Store*
5	playstore_apps	title_play	judul aplikasi pada *Google Play Store*
6	playstore_apps	developer_play	nama perusahaan developer aplikasi pada *Google Play Store*
7	aggregate_apps	is_checked	penanda (*flag*), bernilai 0 jika belum dilakukan proses pencocokan dan bernilai 1 jika sudah
8	aggregate_apps	status	hasil deteksi menggunakan fungsi Levenshtein, bernilai 0 jika cocok (nilai jumlah jarak Levensthein untuk atribut appId, title, dan developer bernilai kurang dari 30) dan bernilai 0 jika tidak
9	aggregate_apps	similartext_status	hasil deteksi menggunakan fungsi Levenshtein dan *Similar Text*

Relasi playstore_apps dan appstore_apps menjadi masukan, sedangkan luaran disimpan pada relasi aggregate_apps. Proses mendeteksi aplikasi yang sama dijabarkan pada Tabel 4.

Table 4. Pseudocode Deteksi Aplikasi yang Sama

	Deskripsi
1	Ambil semua tupel/baris pada playstore_apps (Pi, i=0..n), dengan atribut is_checked = 0, diurutkan dari download terbanyak ke yang paling sedikit.
2	Untuk setiap tupel pada appstore_apps (Aj, j=0..m), konversi semua atribut ke huruf kecil semua (*lowercase*)
3	Hitung jumlah nilai jarak Levenstein (Li), yaitu Li = Levenshtein (Pi(appId_play), Aj(appId_app)) + Levenshtein (Pi(title_play), Aj(title_app)) + Levenshtein (Pi(developer_play), Aj(developer_app))
	Hitung jumlah nilai Similarity Text (Sti), yaitu STi = (similar_text(Pi(appId_play), Aj(appId_app)) + similar_text(Pi(title_play), Aj(title_app)) + similar_text(Pi(developer_play), Aj(developer_app))) / 3
4	Jika Li < 30 maka status = 1; jika tidak, status = 0
	Jika Li < 30 dan STi > 63% maka similartext_status = 1; jika tidak similartext_status = 0
5	Simpan nilai status pada aggregate_apps i(status, similartext_status), dan lakukan update Pi(is_checked) = 1

4.2 Perbandingan Antara Levenshtein Dengan Gabungan Levenshtein dan Similar Text

Dalam menentukan perbandingan, mula-mula ditentukan jumlah sampel data yang diperlukan. Cara menentukannya menggunakan metode slovin dengan toleransi kesalahan 5% dan tingkat presisi 95%. Dalam data ini menggunakan jumlah populasi 2500 data.

$$n = \frac{2500}{1+2500.\ (0.05)^2} \text{x } 100\% \qquad (1)$$

Nilai n didapatkan hasil 344.8 yang dibulatkan menjadi 345 data. Dalam hal ini digunakan 345 sampel data untuk mencakup 2500 data.

Tahapan setelah itu mencari nilai kesalahan di levenshtein dan didapatkan hasil 32 kesalahan, maka untuk mencari nilai keakuratan didapatkan hasil 90.7%.

$$n = \frac{345-32}{345} \text{x } 100 \qquad (2)$$

Lalu untuk mencari nilai kesalahan di gabungan fungsi antara levenshtein dan similar text didapatkan hasil 19 kesalahan, maka untuk mencari nilai keakuratan didapatkan hasil 94.5%

$$n = \frac{345-19}{345} \text{x } 100 \qquad (3)$$

Dari perhitungan tersebut, gabungan fungsi levenshtein dan fungsi similar text lebih baik dengan tingkat akurasi 94.5 %.

4.3 Usability Testing

Pada pengembangan aplikasi, untuk mendapatkan banyak masukan, aplikasi diuji dengan membuat *Minimum Viable Product* (MVP) dari seluruh *use case* yang ada. *Usability testing* dilakukan dengan satu iterasi uji coba. Seluruh hasil dari *usability testing* yang dilakukan didokumentasikan pada dokumen protokol dan hasil uji coba. Pada akhir setiap sesi us*ability testing* yang dilakukan, responden diminta untuk mengisi kuesioner *System Usability Scale* (SUS) yang digunakan untuk menguur kemudahan dari penggunaan aplikasi. Hasil dari kuesioner *System Usability Scale* menunjukkan aplikasi yang dikembangkan memiliki kemudahan dalam penggunaan sebesar 74.5%. Pada kuisioner tersebut menunjukkan bahwa banyak pengguna yang setuju dengan penyataan yang diajukan kepada mereka, seperti yang tertera pada Tabel 5.

Tabel 5. Hasil Justifikasi System Usability Scale

SUS Calculation												
Peserta	Jenis Kelamin	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	Skor SUS
1	Pria	4	2	5	2	4	1	4	2	5	2	82,5%
2	Wanita	4	3	4	2	4	3	4	2	3	2	67,5%
3	Pria	4	2	4	2	2	4	3	2	4	2	62,5%
4	Wanita	5	1	5	2	5	1	5	2	4	5	82,5%
5	Wanita	4	3	4	2	3	2	3	2	4	2	67,5%
6	Pria	4	2	4	2	2	2	4	2	2	4	60,0%
7	Wanita	4	2	5	1	4	1	5	1	5	1	92,5%
8	Pria	4	2	4	4	4	2	4	2	4	2	70,0%
9	Wanita	4	2	5	2	5	2	4	1	4	3	80,0%
10	Wanita	4	2	4	2	4	2	4	1	5	2	80,0%
Total Usabilitas												74,5%

5. SIMPULAN DAN SARAN

Berdasarkan pengerjaan penelitian yang telah dilakukan dapat disimpulkan:

1. Hasil pengambilan data dari *Google Play Store* didapatkan 44.100 aplikasi dan dari *Apple App Store* didapatkan 18.318 aplikasi. Dari kedua toko tersebut, diambil masing-masing 2.500 sampel aplikasi untuk ujicoba agregasi informasi. Agregasi dilakukan dengan mendeteksi aplikasi sama yang berada di kedua toko.

2. Aplikasi yang berada pada kedua toko (*Apple App Store dan Google Play Store*) dideteksi dengan membandingkan kesamaan informasi menggunakan kombinasi fungsi *levenshtein* dan *similar text*. Fungsi *levenshtein* saja dapat menemukan 761 aplikasi dengan keakuratan 90.7%. Kemampuan deteksi ini ditingkatkan dengan penggabungan antara fungsi *levenshtein* dan fungsi *similar text* sehingga dapat

menemukan 1.696 aplikasi dengan keakuratan 94.5%. Sehingga, metode penggabungan dua fungsi diimplementasikan ke prototipe karena hasilnya yang lebih baik.

3. Hasil dari *usability testing* yang dilakukan ke sebanyak 10 responden menunjukkan aplikasi yang dikembangkan memiliki kemudahan dalam penggunaan sebesar 74.5%. Angka ini bermakna cukup baik, namun masih diperlukan pengembangan antarmuka lebih lanjut.

Penggunaan fungsi *Levenshtein* dengan membandingkan per huruf masih kurang akurat untuk menggabungkan sebuah aplikasi sama yang berada di dua toko. Pengembangan berikutnya dilakukan dengan membandingkan blok-blok kata dalam ID aplikasi. Prototipe juga dapat dikembangkan untuk memperkaya metadata aplikasi secara otomatis sehingga meminimalisasi memasukkan data secara manual untuk pengembang aplikasi. Antarmuka prototipe dapat dikembangkan lebih lanjut untuk meningkatkan kepuasan pengguna dalam mengoperasikan dan memenuhi kebutuhannya.

6. UCAPAN TERIMA KASIH

Publikasi ini merupakan bagian dari kegiatan penelitian yang didanai oleh Lembaga Penelitian dan Pengabdian kepada Masyarakat, Institut Teknologi Sepuluh Nopember (LPPM - ITS), dan Kementrian Riset, Teknologi, dan Pendidikan Tinggi; dengan skema Pengabdian kepada Masyarakat Berbasis Penelitian serta Surat Perjanjian Pelaksanaan Pengabdian Masyarakat No: 950/PKS/ITS/2017.

7. DAFTAR RUJUKAN

[1] J. P. M. Kelly Pedotto, Vivey Chen, “The 2016 U.S. Mobile App Report,” 2016.
[2] Gartner, “Gartner Says Mobile App Stores Will See Annual Downloads Reach 102 Billion in 2013.” [Online]. Available: https://www.gartner.com/newsroom/id/2592315. [Accessed: 01-Aug-2017].
[3] P. Roma, F. Zambuto, and G. Perrone, “The role of the distribution platform in price formation of paid apps,” *Decis. Support Syst.*, vol. 91, pp. 13–24, Nov. 2016.
[4] A. Lella and A. Lipsman, “The 2017 U.S. Mobile App Report,” *comScore*, 2017. [Online]. Available: http://www.comscore.com/Insights/Presentations-and-Whitepapers/2014/The-US-Mobile-App-Report. [Accessed: 01-Aug-2017].
[5] H. Malik and E. M. Shakshuki, “Mining Collective Opinions for Comparison of Mobile Apps,” in *Procedia Computer Science*, 2016, vol. 94, pp. 168–175.
[6] D. Cao *et al.*, “Version-sensitive mobile App recommendation,” *Inf. Sci. (Ny).*, vol. 381, pp. 161–175, Mar. 2017.
[7] Y. Liu, L. Liu, H. Liu, X. Wang, and H. Yang, “Mining domain knowledge from app descriptions,” *Journal of Systems and Software*, vol. 133, Elsevier, pp. 126–144, 01-Nov-2017.
[8] N. Genc-Nayebi and A. Abran, “A systematic literature review: Opinion mining studies from mobile app store user reviews,” *J. Syst. Softw.*, vol. 125, pp. 207–219, Mar. 2017.
[9] V. I. Levenshtein, “Binary codes capable of correcting deletions, insertions, and reversals,” *Sov. Phys. Dokl.*, vol. 10, no. 8, pp. 707–710, 1966.
[10] M. Schrenk, *Webbots, spiders, and screen scrapers : a guide to developing Internet agents with PHP/CURL*, 2nd ed. No Starch Press, 2012.
[11] T. M. Connolly and C. E. Beg, *Database systems : a practical approach to design, implementation, and management*, 6th ed. 2015.
[12] J. Brooke, “SUS - A quick and dirty usability scale,” *Usability Eval. Ind.*, vol. 189, no. 194, pp. 4–7, 1996.
[13] A. S. for P. Affairs, “System Usability Scale (SUS),” Sep. 2013.
[14] G. Navarro, “A guided tour to approximate string matching,” *ACM Comput. Surv.*, vol. 33, no. 1, pp. 31–88, 2001.
[15] I. Oliver, *Programming classics: implementing the world’s best algorithms*. Prentice Hall, 1993.

Halaman ini sengaja dikosongkan