Implementasi *Dashboard Reporting Websocket* dengan *Event Regression Test* Berbasis Web Menggunakan Algoritma FIFO

[1] Haris Yusuf Bakhtiar, [2]Saruni Dwiasnati, [3*]Herfandi
Fakultas Ilmu Komputer, Universitas Mercu Buana[1,2]
Fakultas Rekayasa Sistem, Universitas Teknologi Sumbawa[3]

Jl. Raya, RT.4/RW.1, Meruya Sel., Kec. Kembangan, Jakarta, Daerah Khusus Ibukota Jakarta 11650. [2]
Jl. Raya Olat Maras, Batu Alang, Moyo Hulu, Pernek, Kabupaten Sumbawa, Nusa Tenggara Barat. 84371.[1]
41517110072@student.mercubuana.ac.id[1],saruni.dwiasnati@mercubuana.ac.id[2], herfandi@uts.ac.id[3,*],

***Abstract** - Websocket is the new standard for full-duplex communication. The implemented websocket needs to be tested to see the suitability of the functionality created. Testing using the regression test method takes a long time if it is done manually. The integrated websocket automation regression test is considered to be able to solve the current problem. This research will implement websocket event regression test dashboard reporting using the FIFO (First In First Out) algorithm. The result of this research is a dashboard reporting application that is able to make it easier for admins to monitor and retrieve data and is presented in the form of visualizations and graphs that can be analyzed. Automated regression testing simplifies the testing process. The FIFO algorithm is able to process the data queue being tested. Dashboard reporting showing test cases and test scheduling. The results of the first websocket test experiment only opened the connection and did not close the connection to the external websocket, but after the second test everything worked as desired. The dashboard reporting websocket that is made is expected to make it easier for PT. Tokopedia in viewing test results quickly and accurately.*

***Keyword:** Automation Regression Test, Dashboard Reporting, First In First Out Algorithm, Websocket*

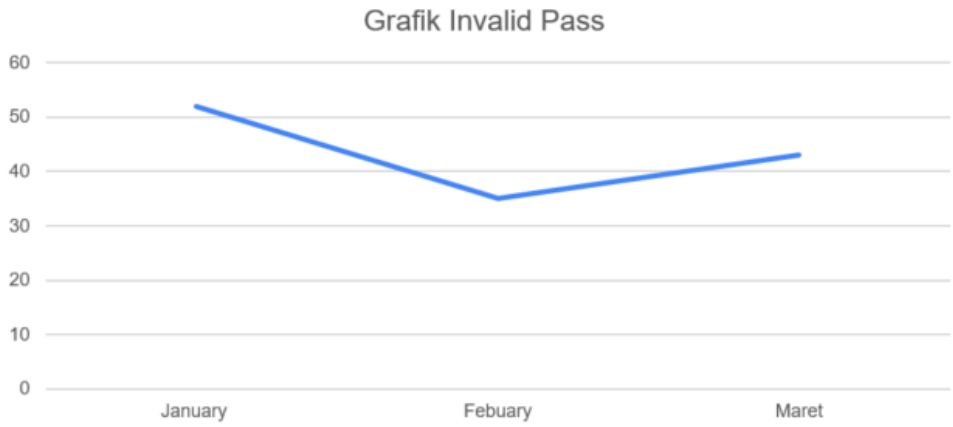
***Abstrak** - **Websocket** merupakan standar baru untuk komunikasi **full-duplex**. **Websocket** yang diimplementasikan perlu untuk dilakukan **testing** guna melihat kesesuaian fungsi yang dibuat. **Testing** menggunakan metode **regression test** membutuhkan waktu yang lama jika dilakukan manual. **Websocket** yang terintegrasi **automation regression test** diduga mampu menyelesaikan masalah yang ada saat ini. penelitian ini akan melakukan implementasi **dashboard reporting websocket event regression test** berbasis web menggunakan algoritma FIFO (**First In First Out**). Hasil dari penelitian ini adalah aplikasi **dashboard reporting** yang mampu memudahkan **admin** dalam melakukan pengawasan dan penarikan data serta disajikan dalam bentuk visualisasi dan grafik yang dapat dianalisis. **Automation regression test** mempermudah proses testing. Algoritma FIFO mampu memproses antrian data yang di testing. **Dashboard reporting** memiliki fitur **test case** dan penjadwalan **testing**. Hasil percobaan pertama **websocket testing** hanya membuka koneksi saja dan tidak melakukan penutupan koneksi ke **websocket** eksternal, akan tetapi setelah pengujian kedua semuanya berjalan sesuai yang diinginkan. **Dashboard reporting websocket** yang dibuat diharapkan mampu mempermuda pihak PT. Tokopedia dalam melihat hasil **testing** secara cepat dan akurat.*

Keyword: ***Automation Regression Test,*** **Algoritma** ***First In First Out, Websocket***

I. PENDAHULUAN

Websocket merupakan standar baru untuk komunikasi *full-duplex* (dua arah secara bersamaan) antara *client* dan *server* [1]. Banyak perusahaan ingin mengimplementasikan *websocket* karena kebutuhan monitoring data secara *realtime* dari *client* ke aplikasi mereka. *Websocket* memiliki kelebihan yaitu mampu menjaga komunikasi *full-duplex* agar tetap terkoneksi dengan baik, memberikan *latency* serta mampu mengurangi kepadatan lalu lintas jaringan yang tidak penting. Berdasarkan observasi dari PT. Tokopedia yang merupakan toko daring dengan model *marketplace*, dalam menjalankan proses bisnisnya selalu menawarkan pelayanan yang berkualitas. Perusahaan inilah yang menjadi tempat studi kasus pada penilitian ini. PT. Tokopedia mengimplementasikan *websocket* pada produk Tokopedia-*Play*. Tokopedia-*Play websocket* digunakan untuk mengirimkan beberapa *event*, dan juga menerima

beberapa *event* dari *client*, contohnya: untuk memperbarui jumlah penonton pada saat *live*. Namun terdapat beberapa masalah yang teridentifikasi saat Tokopedia-*Play websocket* dilakukan *testing* manual selama tiga bulan. Hasil *testing* manual bisa dilihat pada Gambar 1.



Gambar 1 Grafik *Testing* Manual Tokopedia-*Play Websocket*

Grafik pada Gambar 1 menunjukkan hasil yang kurang stabil dari Tokopedia-*Play websocket*, dikarenakan meningkatnya grafik *invalid pass* dari bulan februari ke bulan maret, dimana *testing* manual dilakukan pada jam 09.00 setiap hari kerja. Akan tetapi di jam setelah itu muncul *issue bad deployment* dan konfigurasi *server* bisa berubah saat *autoscale* dilakukan. Hasil Grafik pada Gambar 1 sulit untuk dilakukan analisis yang mendalam dikarenakan penyajian data dengan apa adanya. Karenanya perlu untuk dilakukan monitoring secara *realtime* guna mendukung kualitas pengambilan keputusan proses bisnis yang ada [2]. Oleh sebab itu penelitian ini melakukan implementasi *dashboard reporting websocket* dengan metode tertentu untuk mempermuda admin dalam melihat hasil *testing* secara cepat dan akurat.

Seiring dengan perkembangan pembuatan *software* saat ini, *dashboard reporting* menjadi hal yang wajib dimiliki oleh sebuah web guna mempermudah penyajian data [3]. *Dashboard reporting* merupakan sebuah *user interface* yang berisikan data dan desain dengan menyajikan dalam bentuk berbagai metrik angka ataupun visualisasi data. Keunggulan *dashboard reporting* yaitu bisa menjadi sarana dalam penyajian dan pengolahan data dari sebuah pengujian sebuah fitur, dalam hal ini bisa dimanfaatkan dalam fitur *websocket*. *Websocket* perlu dilakukan *testing* untuk melihat kesesuaian *software* yang di kembangkan berdasarkan tujuan awal dengan harapan bebas dari kecacatan [4]. Ada banyak tipe *testing*, akan tetapi yang mempunyai peranan sangat penting yaitu *regression test*. *Regression test* merupakan tipe pengujian yang berfokus terhadap fungsi dan fitur-fitur sebelumnya yang masih berjalan, dimana *test* ini akan melakukan pengecekan apakah fitur sudah sesuai dengan yang diharapkan pada saat ada fitur yang baru [5]. *Regression test* membutuhkan waktu yang lama jika dilakukan dalam manual [6]. Dipasaran saat ini tidak sedikit *software automation test*, contohnya: *katalon studio, assure-nxt, sigos, Appium Studio, Selenium Studio*. Akan tetapi kebanyakan *software* dipasaran saat ini belum mendukung untuk *websocket automation*. Maka penggunaan *automation* pada *regression test* pada *websocket* akan diterapkan pada penelitian ini.

Websocket event di PT. Tokopedia harus di eksekusi melalui antrian, karenanya algoritma yang di pilih untuk menyelesaikan masalah ini adalah algoritma FIFO (*First In First Out*) [7]. Algoritma FIFO diduga mampu menyelesaikan masalah ini dikarenakan bersifat berurutan dan bergiliran namun tetap pada alur atau jalurnya sesuai dengan yang pertama kali masuk dan kemudian diproses sesuai dengan giliran [8]. Berdasarkan masalah dan referensi yang ada, maka penelitian ini akan melakukan implementasi *dashboard reporting websocket event regression test* berbasis web menggunakan algoritma FIFO (*First In First Out*). *Dashboard reporting* diharapkan mampu memudahkan admin dalam melakukan pengawasan dan penarikan data serta dari hasil pengawasan tersebut dapat disajikan dalam bentuk visualisasi atau grafik yang dapat dianalisis serta data dapat dipertanggungjawabkan. Selain itu data akan disimpan di *database* dan data lebih lama untuk disimpan. *Regression test* digunakan untuk melakukan *testing* pada *websocket* event di PT. Tokopedia dengan fitur *automation* guna mempermudah proses *testing*. Algoritma FIFO (*First In First Out*) digunakan untuk memproses data yang diinputkan pada aplikasi sesuai dengan urutan pertama kali. *Dashboard reporting* ini memiliki fitur *test case*, dengan fitur ini kita bisa dengan mudah menyesuaikan *test cases* apa saja yang akan dijalankan di *websocket* serta fitur penjadwalan *testing* juga mempermudah dalam melakukan *testing* setiap waktu.

Penelitian terkait terdahulu dengan topik yang sama dengan penelitian saat ini adalah sebagai berikut, G. H. Prathama, N. M. Ary Esta Dewi Wirastuti, and Y. Divayana (2019), penelitian ini menggunakan pola *shadow player* dan dapat disimpulkan bahwa *websocket* dan WebRTC memiliki metode pengolahan data serta *websocket* lebih banyak dukungan *browser*. Penelitian terdahulu ini melakukan pengujian manual dan tidak membangun *dashboard reporting* [9], sedangkan pada penelitian ini mengimplementasikan *dashboard reporting* untuk memantau stabilitas,

dengan pengujian *automation* menggunakan *regression test*. K. E. Ogundeyi and C. Yinka-Banjo (2019), artikel pada penelitian ini menggunakan HTTP *Polling* dengan *websocket*. Penelitian ini memiliki kesimpulan bahwa keunggulan *websocket* yaitu *latency* yang kecil, *high-throughput*, lebih sedikit *load network* ke *server*. Penelitian terdahulu ini masih melakukan pengujian manual [10]. Sedangkan pada penelitian saat ini menerapkan *websocket* dengan algoritma FIFO (*First In First Out*) sebagai metode untuk melakukan *regression test* secara *automation* yang berguna untuk memastikan ke stabilan performa dari *websocket*. A. Miu, F. Ferreira, N. Yoshida, and F. Zhou (2020), penelitian ini menggunakan metode MPST *workflow*, hasil penelitian ini menunjukan bahwa *websocket* bisa digunakan menggunakan Bahasa *TypeScript* dengan MPST *Framework* akan tetapi masih ada kekurangan yaitu server harus menggunakan *asynchronous* dalam menerima pesan. penelitian pada artikel ini tidak membangun *dashboard reporting* dan masih melakukan pengujian manual [11]. Sedangkan artikel pada penelitian ini melakukan membangun *dashboard reporting* dengan pengujian *automation* menggunakan *regression test* serta menerapkan algoritma FIFO (*First In First Out*) untuk memproses data yang akan diuji. A. Andrews, A. Alhaddad, and S. Boukhris (2019), penelitian terdahulu ini menggunakan metode MBT web *testing*. Pada penelitian ini *regression testing* digunakan untuk melakukan *testing* pada *fail-safe behavior* di aplikasi berbasis web. *Regression test* digunakan untuk mengklasifikasi *test cases restable*, *reusable* dan *obsolete*. Pada penelitian ini tidak ada algoritma yang diterapkan untuk memproses data antrian [12]. Sedangkan pada penelitian ini menerapkan algoritma FIFO (*First In First Out*) untuk memproses antrian data. Penelitian terkait terdahulu yang berkaitan dengan *websocket*, ada satu masalah yang cukup mengkhawatirkan yaitu stabilitas dari *websocket* itu sendiri. Oleh karena itu dengan adanya *dashboard reporting* ini diharapkan bisa berguna untuk memantau stabilitas dan juga melakukan *automation regression test.* Kebaruan pada penelitian ini adalah menerapkan *websocket* dengan algoritma FIFO (*First In First Out*) sebagai metode untuk melakukan *regression test* secara *automation* yang berguna untuk memastikan ke stabilan performa dari *websocket.*

II. METODE PENELITIAN

Jenis penelitian yang digunakan yaitu metodologi penelitian kuantitatif yang merupakan data sebuah angka [13], yang dimana untuk menghitung nilai dari sebuah hasil *websocket event regression test* untuk membuat suatu *report stability* dan juga notifikasi jika ada yang error atau ada yang tidak sesuai. Data yang digunakan dalam penelitian ini merupakan data yang diambil PT Tokopedia. Adapun tahapan penelitian untuk mengintegrasi *automation websocket* dengan *dashboard monitoring* dapat dilihat pada Gambar 2.

Gambar 2 Tahapan Penelitian

2.1. Identifikasi Masalah

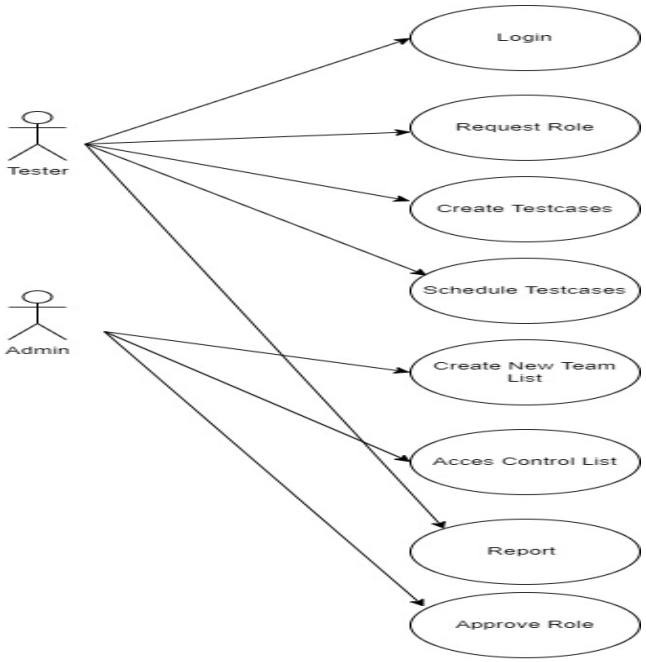
Tahap ini melakukan identifikasi permasalahan yang ada dengan metode observasi [14]. Metode ini bertujuan untuk melakukan pengamatan secara langsung untuk mengetahui bagaimana cara kerja dan alur *testing websocket event* serta mengetahui *report* yang dibutuhkan untuk laporan *websocket*. Berdasarkan data yang sudah diamati dari hasil *testing* menunjukkan bahwa *websocket* dalam kondisi tertentu akan tidak stabil ini menyebabkan tidak validnya hasil *testing* manual yang dilakukan. Oleh karena itu untuk mengidentifikasi masalah lebih cepat lagi diperlukannya *automation testing* yang bisa dijalankan kapanpun dan dibutuhkan juga dashboard reporting yang digunakan sebagai pengolahan data dan penyajian data.

2.2 Studi Literatur

Tahap studi literatur dilakukan untuk memperoleh dasar teori sebagai referensi dalam penelitian. Pengumpulan data dan pemahaman berbagai referensi yang berkaitan dengan topik penelitian yang dipilih guna menunjang pembuatan pembuatan sistem. Referensi yang dikumpulkan berupa jurnal, buku, dan *e-book* [15].

2.3 Perancangan

Tahap yang ketiga digunakan untuk merancang sebuah solusi yang dapat mengatasi masalah yang sudah teridentifikasi pada metode sebelumnya. Perancangan ini dimaksudkan untuk menentukan perancangan *dashboard* yang cocok untuk meng-akomodasi dari kebutuhan untuk pengawasan dan melakukan *reporting*. Perancangan yang dilakukan menghasil dua keluaran yaitu perancangan sistem dan perancangan UI. Metode yang digunakan untuk perancangan sistem adalah UML (*Unified Modelling Language*) yaitu suatu metode pemodelan secara visual untuk sarana perancangan sistem berorientasi objek [16]. *Use Cases Diagram* bertujuan untuk menggambarkan interaksi antar aktor dengan sistem. *Use Case Diagram dashboard reporting websocket* dapat dilihat pada Gambar 3.



Gambar 3 *Use Case Diagram Dashboard Reporting Websocket*

Seperti yang ditunjuk pada Gambar 3, dapat di deskripsikan bahwa terdapat dua aktor yang dapat melakukan interaksi dengan sistem yaitu *tester* dan *admin*. Aktor *tester* dan *admin* harus melakukan *login* sebelum menggunakan sistem. Aktor *tester* dapat melakukan *request role* yang digunakan untuk melakukan *action* didalam aplikasi, melakukan *create test cases* atau pembuatan *test cases*, melakukan *schedule test cases* atau melakukan penjadwalan pada running *test cases*, serta melakukan *report* atau bisa disebut juga untuk mengambil *report*. Aktor *admin* dapat melakukan *create new team list* atau membuat daftar *team* baru yang bisa digunakan untuk *request*, melakukan *access control list* atau melakukan administrasi pada *role user*, sarta dapat melakukan *approve role*.

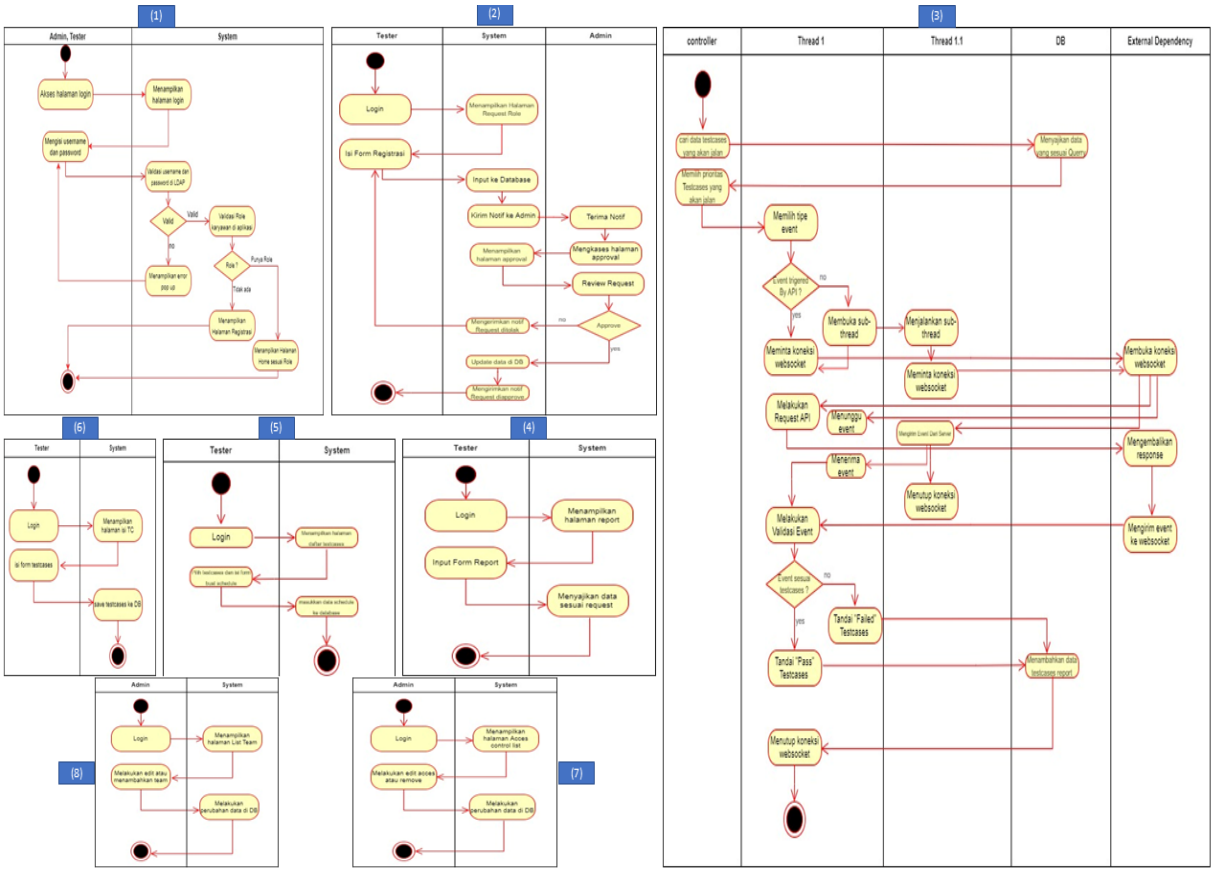
2.4 Implementasi dan Pengujian

Dari hasil perancangan peneliti melakukan implementasi agar bisa dijalankan secara tepat. Keluaran dari hasil metode ini adalah *dashboard* yang sudah siap digunakan dan bisa diisi oleh data dan berjalan dengan sesuai harapan. Pembuatan *dashboard* ini menggunakan *Golang* sebagai vahasa *BackEnd*-nya, *FrontEnd*-nya menggunakan PHP dan JS, *Database* Menggunakan MySQL, untuk komunikasi antar *platform* menggunakan REST API. Pengujian yang dilakukan pada penelitian ini dengan cara melakukan hit endpoint untuk menjalankan runner setelah itu sistem akan membaca data *test casese* dari *database* setelah itu maka akan keluar *log* di sistem menunjukan *hit endpoint* tersebut berjalan seperti gambar berikut.

III. HASIL DAN PEMBAHASAN

3.1 *Activity Diagram*

Activity Diagram dibuat untuk menggambarkan aliran kerja atau aktivitas dari sebuah aplikasi atau proses bisnis yang harus dilakukan pada aplikasi oleh aktor. *Activity diagram* pada aplikasi disajikan pada Gambar 4.



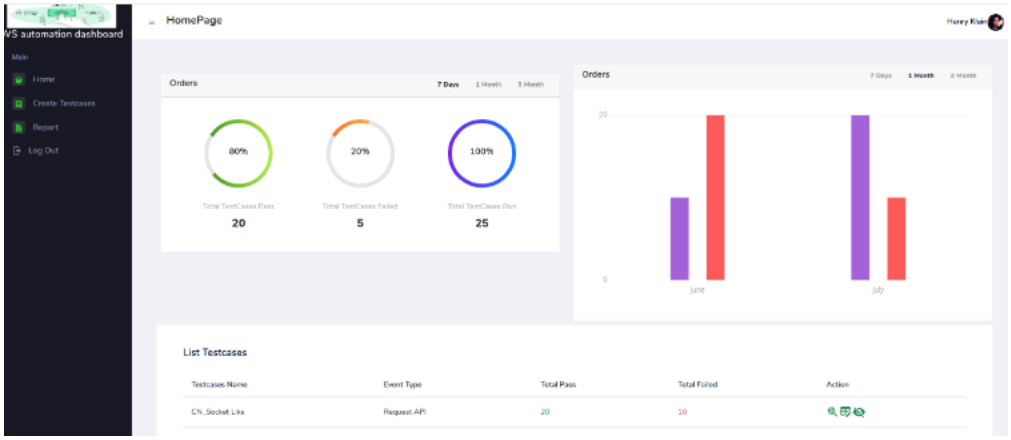
Gambar 4 *Activity Diagram* pada Aplikasi

Deskripsi Gambar 4 sebagai berikut, (1) *Activity Diagram Login*: *admin* dan *tester* melakukan *login* pada sistem, kemudian sistem memverifikasi data yang telah diinput, jika data benar dan *valid* maka *user* akan menuju pada validasi selanjutnya yaitu *validasi role* disini akan ditentukan *user* tersebut sudah memiliki *role* atau tidak jika sudah memiliki *role* akan diarahkan menuju halaman *role* masing-masing akan tetapi jika tidak sesuai maka akan melakukan *request role*. (2) *Activity Diagram Request Role*: *tester* melakukan *request role*. Jadi tahapan yang dilakukan oleh aktor *tester* adalah mengisi *form* registrasi, setelah selesai maka sistem akan melakukan proses penyimpanan di *database* dan setelah itu *request* akan diverfikasi dan jika *request* di *approve* maka akan mengirimkan *notif* kepada *tester* jika tidak maka *tester* akan melakukan *request* ulang jika ada beberapa catatan. (3) *Activity Diagram Schedule Test Cases*: *controller* dalam menjalankan proses untuk mengeksekusi *test cases*. Tahapan yang dilakukan adalah *controller* akan meminta data ke *databases* mana saja *test cases* yang akan dijalankan maka *controller* akan mengurutkan *test cases* mana saja berdasarkan tanggal *test cases* dibuat, dan prioritas. Setelah itu sistem akan memilih tipe *event* jika *event* di kirim setelah melakukan *request* API maka sistem akan menjalankan *request* API terlebih dahulu setelah itu melakukan validasi terhadap *event* yang dikirimkan. Akan tetapi jika event dikirimkan secara langsung oleh *server* maka sistem akan melakukan simulasi pengiriman dengan cara membuka proses baru atau *thread* baru yang akan bertugas sebagai pengiriman, jika proses pengiriman sudah selesai maka proses utama akan memverifikasi hasil yang dikirimkan oleh *server* tersebut. (4) *Activity Diagram Generate Report*: *tester* akan melakukan *input form* yang digunakan untuk meng *genarate* data dan sistem akan menyajikan data sesuai *request*. (5) *Activity Diagram Schedule Test Cases*: *tester* melakukan *login*, memilih *test cases* yang akan di jadwalkan dan mengisi *form* untuk penjadwalannya. Setelah itu sistem akan melakukan penyimpanan konfigurasi penjadwalan pada *database*. (6) *Activity Diagram Create Test Cases*: *tester* melakukan isi *form test cases* setelah itu langsung di save ke *Database* oleh sistem. (7) *Activity Diagram Access Control List*: *tester* akan melakukan *edit access* atau *remove* setelah itu sistem akan melakukan perubahan data dalam *Database*. (8) *Activity Diagram Create*: *admin* akan melakukan *edit* atau menambahkan daftar *admin* dan sistem yang akan melakukan perubahan data di *Database*.

3.2 Implementasi Antarmuka *Homepage Tester*

Tampilan antarmuka *homepage tester* yang disajikan pada Gambar 5 menunjukan grafik total *test cases pass*, total *test cases failed* dan total *test cases* yang sudah dijalankan dibandingkan dengan total *test cases* yang sudah dibuat. Selain itu juga ada grafik yang menunjukkan *tren* dari pengawasan *websocket* tersebut dengan *range* waktu

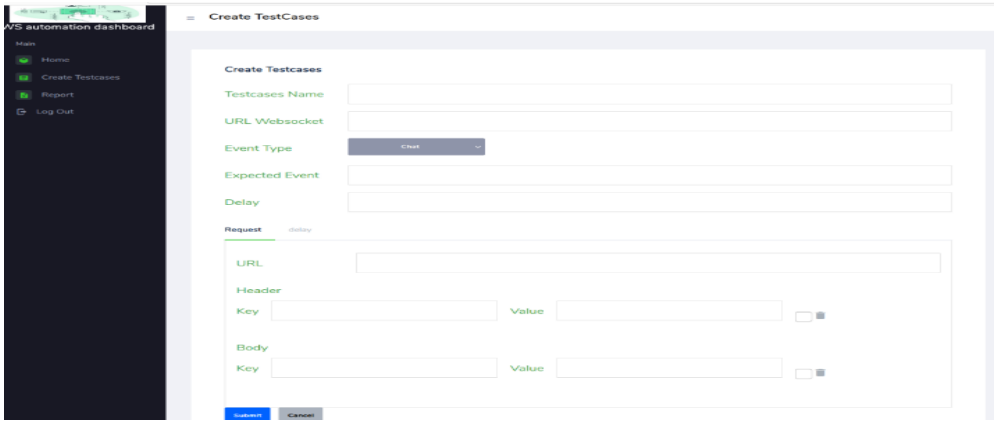
yang disediakan mulai dari minggu, bulan dan dalam satu tahun. Dan tak lupa tentang *table test cases* yang sudah dibuat dengan dilengkapi beberapa *action detail*, *edit*, dan *hide* atau tidak dijalankan terlebih dahulu.



Gambar 5 Antarmuka *Homepage Tester*

3.7 Implementasi *Create Test Cases*

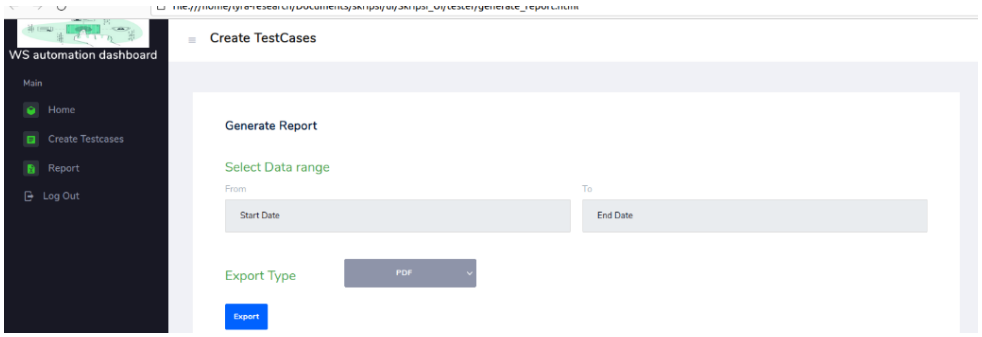
Tampilan halaman yang bi dilihat pada Gambar 6, *tester* bisa menentukan, bagaimana *websocket* itu diuji untuk mengetahui kestabilannya dalam tampilan ini juga tesiter bisa menentukan parameter atau indikasi bahwa *websocket* tersebut sudah berjalan dengan baik dengan menerima *event* yang sudah ditentukan dan bagaimana *event* tersebut akan dimunculkan apakah melalui *request* API atau dengan cara dari *server* akan mengirimkan langsung contoh *chat*.



Gambar 6 Tampilan Halaman *Create Test Cases*

3.8 Implementasi *Generate Report*

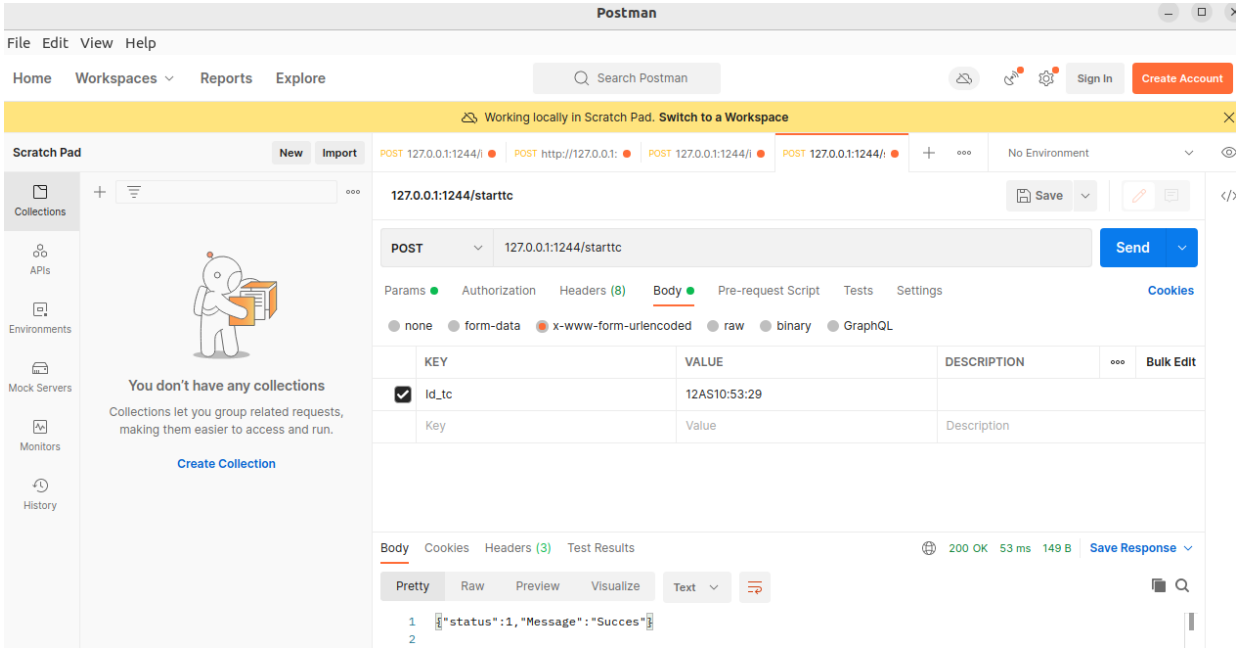
Pada halaman yang bisa dilihat pada Gambar 7, *tester* bisa melakukan *export* terhadap hasil *websocket* yang bisa diolah melalui aplikasi lain, yaitu dalam bentuk *spreedsheet*, ataupun PDF. Untuk halaman ini *tester* akan memilih kapan data akan dibuatkan *reportnya*, dengan hanya mengisi *start date* dan *end date* maka akan secara otomatis sistem akan membuatkan reportnya.



Gambar 7 Tampilan Halaman *Create Test Cases*

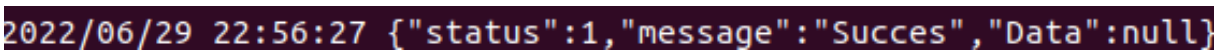
3.9 Pengujian dengan Eksperimen

Tahapan Eksperimen terbagi menjadi 4 bagian: *Hit Rest API*, Cek *Log* sistem jika berjalan,Cek *Client* lain apabila sistem melakukan koneksi ke *websocket*, Cek *database* apakah data hasil sudah masuk. Langkah pertama kita akan menjalankan lewat REST API dengan inputan *Id test cases* yang mau dijalankan dengan tipe data adalah *x-www-form-urlencoded* setelah semua inputan kita isi maka kita akan melakukan request yang bisa dilihat pada Gambar 8.



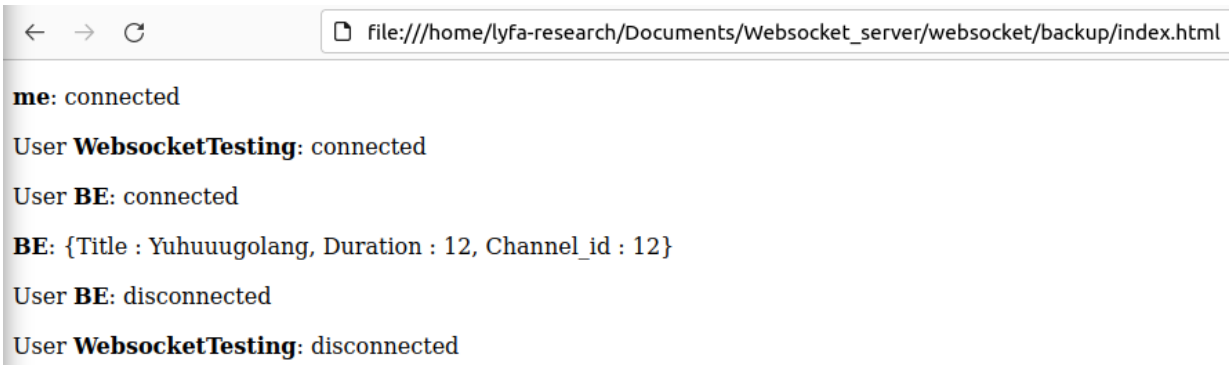
Gambar 8 *Hit* REST API

Langkah kedua kita akan melakukan cek pada *log* sistem apakah sistem berjalan dengan normal atau ada kesalahan sistem dengan mengecek *log* yang terjadi jika ada kesalahan pada sistem, langka ini disajikan pada Gambar 9.



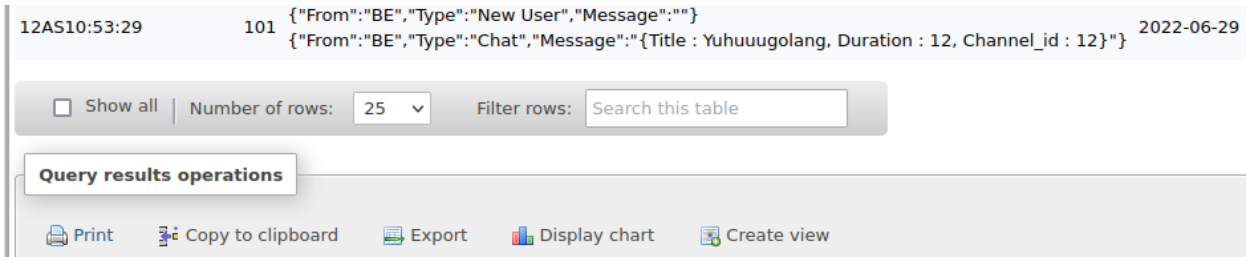
Gambar 9 *Log* Sistem

Gambar 10 menunjukkan bahwa sistem kita yang diwakili dengan *user websocket* Testing sudah sukses melakukan koneksi ke *websocket* baru setelah itu muncul sebuah user BE yang mewakili dari *eksternal website* setelah itu BE mengirimkan *event* berisi data *title duration* dan *channel id*. Setelah *backend* sukses mengirim *event* maka akan melakukan *close* koneksi ke *client* dan yang selanjutnya Sistem kita akan melakukan *close* koneksi juga.



Gambar 10 *Web Client*

Gambar 11 menunjukkan bahwa sistem sudah menyelesaikan pengujian dan mengirim hasilnya ke *database*, dengan informasi bahwa status *test cases* adalah 101 yang menunjukkan kode *passed*, dengan *log* yang diterima dari BE seperti yang sudah ditunjukkan.



Gambar 11 *Database*

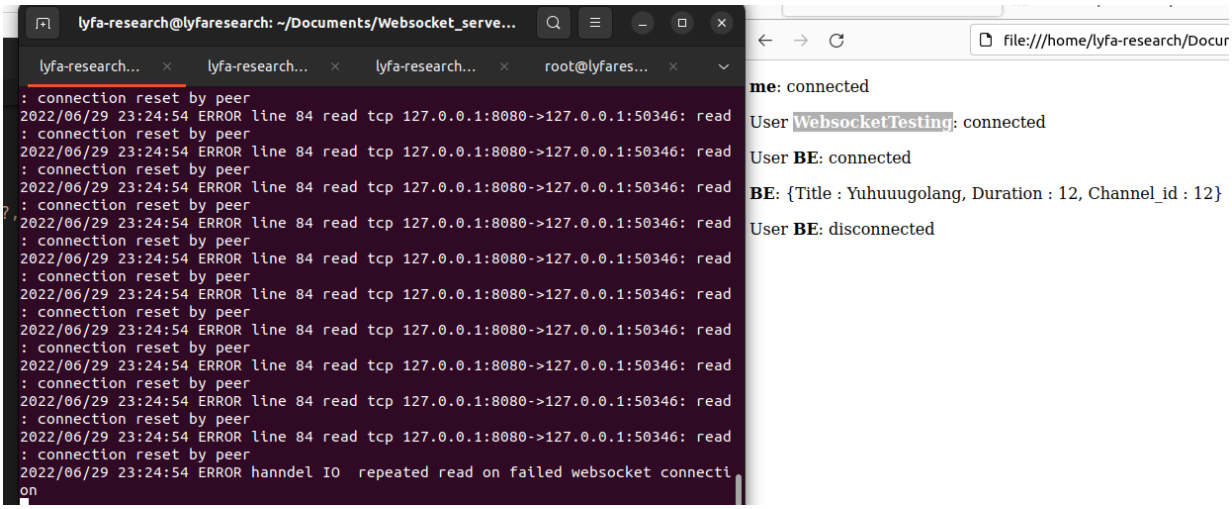
3.9 Hasil Pengujian dengan Eksperimen

Skenario pada Tabel 1 didapat *websocket* sempat tidak tertutup dengan baik hal ini sangat berbahaya bagi sistem eksternal yang dapat menyebabkan antrian koneksi yang sangat banyak oleh karena itu perlu adanya pengecekan ulang pada kode penutupan koneksi. Berikut bukti *websocket* sistem pengecekan tidak tertutup.

Tabel 1 Hasil Skenario Positif Percobaan 1

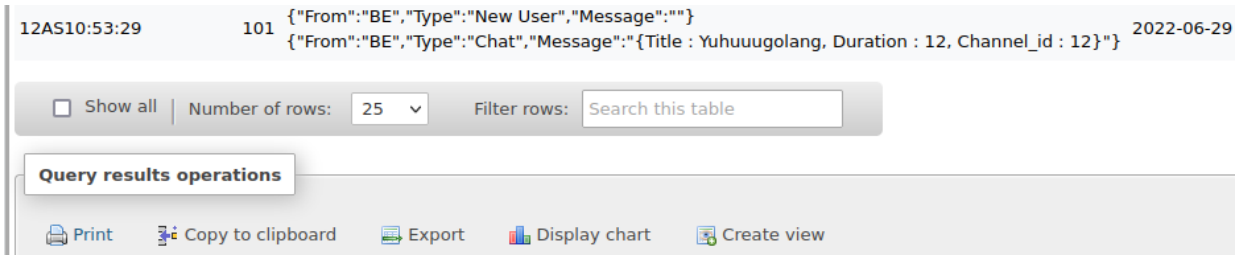
Skenario	
Sebagai *user* saya dapat menjalankan websocket automation regression, setelah pengujian selesai *websocket* harus menutup koneksi	
Hasil yang diinginkan	**Hasil yang didapat**
Sistem pengecekan berjalan	Sistem pengecekan berjalan
Websocket pengecekan terkoneksi	*Websocket* pengecekan terkoneksi
Websocket pengecekan tertutup	***Websocket* pengecekan tidak tertutup**
Sistem sukses untuk memasukkan data dari *database*	Sistem sukses untuk memasukkan data dari *database*

Bisa terlihat pada Gambar 12 bahwa *websocket* Testing hanya membuka koneksi saja tidak melakukan penutupan koneksi seperti yang BE (*websocket* eksternal) lakukan, padahal data di database sudah masuk dan terjadi penumpukan *error* di *log websocket*.

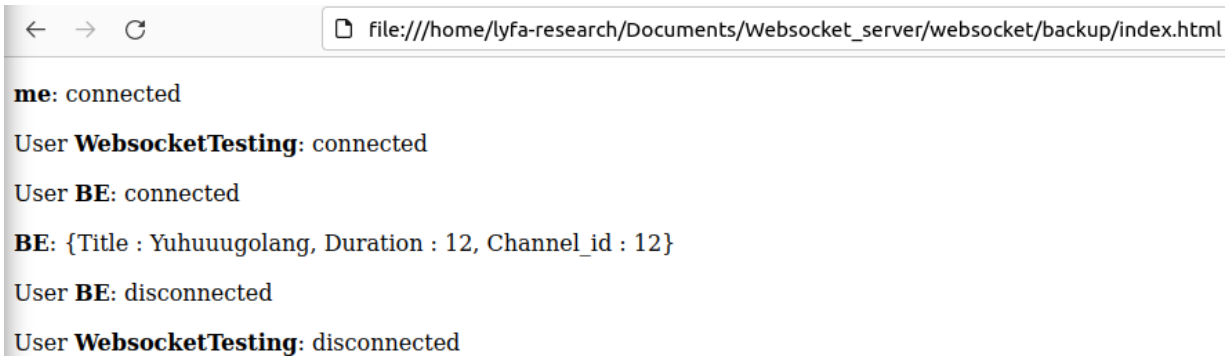


Gambar 12 Gagal untuk Menutup Koneksi *Websocket*

Hasil Skenario Positif Percobaan 2 dilakukan sesuai dengan Tabel 1 akan tetapi kali ini hasil yang didapatkan adalah ***Websocket* pengecekan tertutup,** sesuai dengan hasil yang diinginkan. Data hasil pengecekan masuk ke *database* bisa dilihat pada Gambar 13, sedangkan data hasil pengecekan *client* lain tersaji pada Gambar 17.



Gambar 13 Data Hasil Pengecekan Masuk ke *Database*



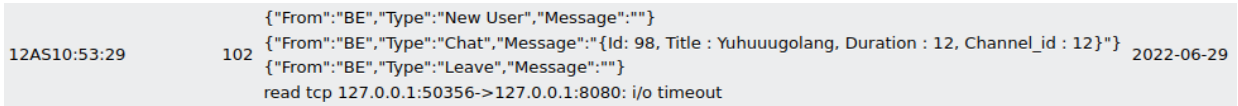
Gambar 14 Data Hasil Pengecekan *Client* lain.

Skenario pada Tabel 2 kita mencoba untuk mensimulasikan jika terjadi kasus *websocket event* ada perubahan dari eksternal secara tiba-tiba, dan yang terjadi adalah sistem mendeteksi ketidaksesuaian. Berikut hasil yang didapat dan *report* ke *database*.

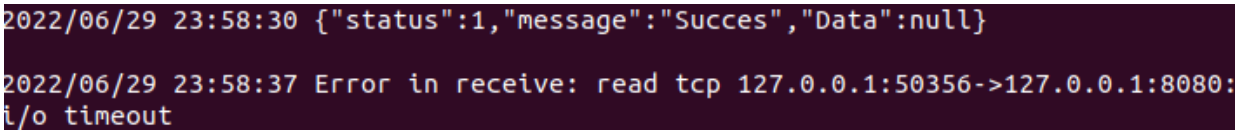
Tabel 2 Hasil Skenario Negatif

Skenario	
Sebagai *user* saya dapat menjalankan *websocket automation regression*, setelah pengujian selesai *websocket* harus menutup koneksi	
Hasil yang diinginkan	**Hasil yang didapat**
Sistem pengecekan berjalan	Sistem pengecekan berjalan
Websocket pengecekan terkoneksi	*Websocket* pengecekan terkoneksi
Websocket pengecekan tertutup	*Websocket* pengecekan tidak tertutup
Sistem sukses untuk memasukkan data dari *database* dengan status gagal	Sistem sukses untuk memasukkan data dari *database* dengan status gagal

Hasil dari Gambar 15 dan 16 menjelaskan bahwa data yang diterima sistem pengecekan tidak sesuai sehingga sistem akan menjalankan *procedure* untuk menunggu terlebih dahulu. Jika sistem sudah menunggu untuk berapa waktu maka sistem akan mematikan paksa dan memberikan tanda bahwa *event* tidak diterima sehingga menghasilkan *test cases* gagal.



Gambar 15 Data *Report* dari *Database*



Gambar 16 Data *Log* dari sistem

IV. KESIMPULAN

Berdasarkan hasil dari analisis pengujian serta pembahasan yang telah dilaksanakan. Memberikan hasil bahwa dimungkinkan untuk mengintegrasikan *websocket* automation dengan *dashboard reporting*, meskipun dalam praktiknya masih ada beberapa yang perlu diperbaiki lagi tentang pengintegrasian ini. Oleh karena itu dapat

disimpulkan bahwa dengan adanya implementasi integrasi *websocket automation regression test* akan memudahkan dalam menjaga stabilitas *websocket* dengan penyajian data dalam berbagai bentuk. Selain itu *dashboard* yang diimplementasikan menyediakan *custom cases* untuk berbagai macam kebutuhan evaluasi dari *websocket* tersebut.

Penelitian yang dilakukan saat ini perlukan dilakukan tinjauan lebih lanjut kedepannya. Karenanya diharapkan untuk peneliti selanjutnya bisa menerapkan metode dan *tools* yang berbeda guna memberikan kontribusi yang lebih baik lagi dari penelitian yang saat ini dilakukan.

REFERENSI

[1] A. H. al Baaits and W. K. Raharja, "Perancangan dan Simulasi Proses Antrean Data Multisensor Untuk Sistem Telemonitoring Multikontrol Berbasis Internet of Things," *Teknika*, vol. 11, no. 1, pp. 53–61, Mar. 2022, doi: 10.34148/teknika.v11i1.418.

[2] S. Salsabila, A. Trisnadoli, and I. Muslim, "Rancang Bangun Sistem Informasi Monitoring Menggunakan Metode Agile dengan Dynamic System Development Model Guna Mendukung Gender Mainstreaming Strategy (Studi Kasus: Politeknik Caltex Riau)," *TEKNIK*, vol. 40, no. 3, p. 195, Dec. 2019, doi: 10.14710/teknik.v40i3.25704.

[3] H. Herfandi, E. Ramdani, S. Dwiasnati, A. Susilo Yuda Irawan, and R. Habibie Sukarna, "Penerapan Self-Service Technology pada Aplikasi Pelayanan Penduduk Desa Labuhan Sumbawa," *Format : Jurnal Ilmiah Teknik Informatika*, vol. 11, 2022, doi: http://dx.doi.org/10.22441/10.22441/format.2022.v11.i1.010.

[4] H. Nurwarsito and R. D. Christian, "River Water Pollutant Level Monitoring System using Websocket Protocol and LoRa Communication Module," in *2021 2nd International Conference on ICT for Rural Development (IC-ICTRuDev)*, Oct. 2021, pp. 1–6. doi: 10.1109/IC-ICTRuDev50538.2021.9656506.

[5] W. Lam, R. Oei, A. Shi, D. Marinov, and T. Xie, "iDFlakies: A Framework for Detecting and Partially Classifying Flaky Tests," in *2019 12th IEEE Conference on Software Testing, Validation and Verification (ICST)*, Apr. 2019, pp. 312–322. doi: 10.1109/ICST.2019.00038.

[6] A. A. Elhadidy, S. M. El-Badawy, and E. E. Elbeltagi, "A simplified pavement condition index regression model for pavement evaluation," *International Journal of Pavement Engineering*, vol. 22, no. 5, pp. 643–652, Apr. 2021, doi: 10.1080/10298436.2019.1633579.

[7] M. N. Nababan, W. Purba, and E. Indra, "The Tuition Payment Queuing System Uses Android-Based First in First Out (FIFO) Algorithm," *IOP Conference Series: Earth and Environmental Science*, vol. 748, no. 1, p. 012038, Apr. 2021, doi: 10.1088/1755-1315/748/1/012038.

[8] Y. Zhang *et al.*, "DepGraph: A Dependency-Driven Accelerator for Efficient Iterative Graph Processing," in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, Feb. 2021, pp. 371–384. doi: 10.1109/HPCA51647.2021.00039.

[9] G. H. Prathama, N. M. Ary Esta Dewi Wirastuti, and Y. Divayana, "Analisa Penggunaan WebRTC dan Websocket pada Real Time Multiplayer Online Game Tradisional Ceki," *Majalah Ilmiah Teknologi Elektro*, vol. 18, no. 1, p. 47, May 2019, doi: 10.24843/MITE.2019.v18i01.P07.

[10] K. E. Ogundeyi and C. Yinka-Banjo, "WebSocket in real time application," *Nigerian Journal of Technology*, vol. 38, no. 4, p. 1010, Dec. 2019, doi: 10.4314/njt.v38i4.26.

[11] A. Miu, F. Ferreira, N. Yoshida, and F. Zhou, "Generating Interactive WebSocket Applications in TypeScript," *Electronic Proceedings in Theoretical Computer Science*, vol. 314, pp. 12–22, Apr. 2020, doi: 10.4204/EPTCS.314.2.

[12] A. Andrews, A. Alhaddad, and S. Boukhris, "Black-box model-based regression testing of fail-safe behavior in web applications," *Journal of Systems and Software*, vol. 149, pp. 318–339, Mar. 2019, doi: 10.1016/j.jss.2018.11.020.

[13] S. Humble, *Quantitative Analysis of Questionnaires*. Routledge, 2020. doi: 10.4324/9780429400469.

[14] H. Herfandi, E. Purwirawansyah, A. S. Yuda irawan, and K. A. Baihaqi, "Rancang Bangun Alat Pendeteksi Kebocoran Liquefied Petroleum Gas Berbasis Mikrokontroler Wemos D1 R1 Dengan Notifikasi Calling," *Voteteknika (Vocational Teknik Elektronika dan Informatika)*, vol. 10, no. 2, p. 42, Jul. 2022, doi: 10.24036/voteteknika.v10i2.118061.

[15] R. Avrizal and S. Dwiasnati, "Implementasi Data Mining dengan Algoritma C4.5 pada Penjualan Obat," *Format : Jurnal Ilmiah Teknik Informatika*, vol. 8, no. 1, p. 77, Aug. 2019, doi: 10.22441/format.2019.v8.i1/010.

[16] S. Sundaramoorthy, *UML Diagramming*. Boca Raton: Auerbach Publications, 2022. doi: 10.1201/9781003287124.