LAPORAN PENELITIAN

INTEGRATED WEB TEST ENVIRONMENT

Disusun Oleh:

Gede Karya, S.T., M.T.

Elisati Hulu, S.T., M.T.

Lembaga Penelitian dan Pengabdian kepada Masyarakat

Universitas Katolik Parahyangan

Desember 2012

Kata Pengantar

Puji syukur penulis sampaikan kepada Tuhan Yang Maha Pengasih, atas berkah dan bimbingannya, penulis dapat merampungkan laporan penelitian ini. Laporan penelitian ini merupakan rekapitulasi dari sejumlah aktivitas untuk manghasilkan sebuah perangkat lunak berbasis web untuk mengelola otomasi uji fungsional dan uji performansi dan kapasitas web enterprise. Perangkat lunak ini diperlukan untuk memudahkan proses *quality assurance* terutama dalam menguji program-program aplikasi berbasis web sehingga meningkatkan produktifitas dan kualitas perangkat lunak yang dihasilkan. Penelitian ini didanai oleh Unpar melalui LPPM.

Terima kasih kepada Ketua Jurusan Teknik Informatika, Dekan Fakultas Teknologi Informasi dan Sains (FTIS) atas dukungan dan arahan serta persetujuan atas usulan penelitian ini. Terima kasih kepada Wakil Dekan 1 FTIS atas dukungan dan fasilitasinya dalam seminar baik proposal maupun progress/ hasil penelitian. Juga kepada Ketua LPPM atas pendanaan yang diberikan. Selain itu juga Penulis sampaikan terima kasih kepada pak Elisati Hulu atas kerjasamanya dalam beberapa penelitian ini. Kepada para mahasiswa yang telah membantu dalam proses implementasi dan pengujian program serta dokumentasi, diantaranya: M. Aditya Nugraha, M. Iqbal, Ari Bratasena, Nurisya Hafizah dan Yurpita Zey, kami mengucapkan banyak terima kasih.

Akhir kata semoga laporan ini dapat memberikan gambaran motivasi, metodologi dan hasil dari penelitian yang telah dilakukan.

Ketua Peneliti,

Gede Karya, S.T., M.T.

Abstrak

Penelitian ini membahas tentang bagaimana mengintegrasikan sistem uji performansi dan kapasitas (beban) dengan memanfaatkan *distributed multi agent* dengan sistem otomasi uji fungsional (*functional test automation*) web enterprise pada 3 penelitian sebelumnya menjadi *integrated web test environment*. Integrasi dilakukan dengan menggunakan prinsip federasi melalui antarmuka pengguna berbasis web. Fitur yang ditambahkan adalah pengelolaan user, proyek dan testing. Fitur-fitur eksisting seperti: pengelolaan agen, skenario, task, order dan hasil uji diadopsi dan diadaptasi dari antarmuka web otomasi uji fungsional. Penelitian ini menghasilkan suatu situs uji web terintegrasi dengan alamat http://webtest.unpar.ac.id. Berdasarkan hasil uji fungsional, bahwa situs ini telah berfungsi sesuai dengan spesifikasi sebagai *integrated web test environment*. Selanjutnya direkomendasikan untuk digunakan sebagai alat uji substantif pada audit web enterprise, dan pengujian-pengujian performansi dan kapasitas web dalam berbagai konfigurasi, seperti: *server farm* (multi server) dan *cloud web service*.

Kata kunci: *integrated web test environment, functional test automation, distributed multi agent*

Daftar Isi

Daftar Gambar

BAB 1 PENDAHULUAN

Pada bab ini akan dijelaskan latar belakang, rumusan masalah, tujuan dan hasil serta sistematika pembahasan.

1.1. Latar Belakang

Salah satu hal terpenting dalam sebuah produk perangkat lunak adalah kualitas. Kualitas didefinisikan sebagai kemampuan suatu produk untuk memenuhi/ memuaskan kebutuhan penggunanya [1]. Kebutuhan perangkat lunak dapat dibagi menjadi 2, yaitu: kebutuhan fungsional dan kebutuhan non fungsional. Kebutuhan fungsional berkaitan dengan proses bisnis atau fungsi yang diotomasi oleh perangkat lunak. Fungsi ini terkait dengan aktifitas dan tata cara/ perhitungan (alur logika) yang ada di dalamnya beserta data-data yang dioleh oleh perangkat lunak tersebut. Kebutuhan non fungsional mencakup kebutuhan akan reliabilitas, kapasitas, performansi dan sejenisnya.

Bagaimana cara memastikan bahwa suatu perangkat lunak berkualitas? Caranya adalah dengan melakukan uji kualitas (pengujian). Karena komponen kualitas adalah fungsional dan non fungsional, maka pengujiannyapun demikian. Pada pengujian fungsional, dapat dilakukan uji terima yaitu dengan mengecek semua fungsi yang disyaratkan apakah sudah dapat berjalan/ diakomodasi oleh perangkat lunak. Jadi fungsi perangkat lunak dibandingkan dengan fungsi-fungsi dalam proses bisnis yang diotomasi. Dengan demikian dapat dilakukan secara simulasi pada lingkungan pengembangan.

Khusus untuk uji non fungsional, ada beberapa yang tidak dapat dilakukan dengan mudah, misalnya: uji performansi pada saat *peak season*. Ini memerlukan lingkungan ekstrem sesuai dengan definisi kapasitas, berupa *sample user* dan kondisi transaksi maksimal. Pengujian seperti ini juga disebut *load test* atau *stress test.* Demikian juga dengan uji reliabilitas, memerlukan sejumlah kapasitas dan rentang waktu yang cukup panjang untuk menjamin bahwa dalam jangka waktu yang disyaratkan perangkat lunak tidak akan bermasalah.

Untuk suatu kelayakan operasi, baik uji fungsional maupun uji non fungsional harus dilakukan, sehingga menjamin bahwa perangkat lunak sudah siap. Hal ini menjadi suatu resiko yang signifikan khususnya pada sistem enterprais, di mana akan melayani pengguna dan transaksi dalam jumlah besar dengan intensitas dan reliabilitas yang memadai. Khusus untuk uji non fungsional, diperlukan suatu alat dan lingkungan uji yang dapat mewakili kondisi yang ingin diujikan.

Pada 2 penelitian sebelumnya [2][3] sudah dihasilkan *tools* untuk melakukan uji coba secara non fungsional berupa uji performansi/ kinerja dan kapasitas situs berbasis web. Pada penelitian 3 [4] juga telah dikembangkan tools untuk melakukan uji fungsional. Pada penelitian ini dibahas khusus bagaimana melakukan integrasi perangkat lunak uji tersebut, dengan menambahkan antarmuka berbasis web, sehingga mudah dioperasikan oleh penguji (*tester*).

1.2. Rumusan Masalah dan Batasan

Beberapa masalah yang ingin dijawab dalam penelitian ini adalah:

> Bagaimana mengintegrasikan perangkat lunak uji kapasistas dan performansi dengan perangkat lunak uji fungsional dengan antarmuka berbasis web?

Batasan yang diterapkan pada penelitian ini adlaah:

1. Integrasi dilakukan dengan meminimasi perubahan perangkat lunak ekisting.
2. Antarmuka dibuat sesederhana mungkin dan mirip dengan antarmuka web pada aplikasi uji fungsional, dengan demikian, tester merasa ini adalah pengembangan dari perangkat lunak tersebut.
3. Pengujian dibatasi pada pengujian fungsional untuk meyakinkan semua fungsi dari aplikasi integrator telah berjalan dengan baik.

1.3. Tujuan dan Hasil yang Diharapkan

Berdasarkan latar belakang dan rumusan masalah di atas, maka tujuan yang ingin dicapai pada penelitian ini untuk mengembangkan perangkat lunak berbasis web sebagai antarmuka yang mengintegrasikan uji fungsional dan uji performansi/ kapasitas web enterprise.

Oleh karena itu, hasil akhir yang diharapkan dalam penelitian ini adalah sebuah perangkat lunak berbasis web dengan fungsi sebagai berikut:

1. Pengelolaan user (tester).
2. Pengelolaan proyek.
3. Pengelolaan testing.
4. Pengelolaan agen uji (*agent*).
5. Pengelolaan skenario dan task (detail), berikut atribut-atributnya.
6. Pengelolaan pesanan pengujian (order).
7. Pengelolaan hasil uji (*result*).

1.4. Metodologi Penelitian

Penelitian ini dilakukan dengan metode rekayasa produk, khususnya produk perangkat lunak. Tahapan yang dilalui antara lain:

1. Studi Referensi. Diawali dengan refine dari 3 penelitian sebelumnya terkait dengan uji kapasitas dan peformansi serta uji fungsional dengan otomasi multi agen. Kemudian dilanjutkan dengan studi perangkat lunak pembanding/ referensi. Dalam hal ini diambil 3 perangkat lunak, yaitu: Jmeter, Selenium dan Watir.

2. Rekayasa perangka lunak, dengan tahapan sebagai berikut:

    a. Analisis kebutuhan perangkat lunak, baik fungsional maupun non fungsional.

    b. Disain perangkat lunak, mulai dari disain global dan disain detail (basis data, dan aplikasi).

    c. Implementasi, berupa pengkodean menggunakan PHP.

    d. Pengujian, yang dibatasi pada pengujian fungsional dengan menjalankan beberapa kasus uji. Yang difokuskan adalah apakah perangkat lunak integrator ini sudah dapat berfungsi sesuai spesifikasi.

1.5. Sistematika Pembahasan

Laporan penelitian ini disajikan dengan sistematika sebagai berikut:

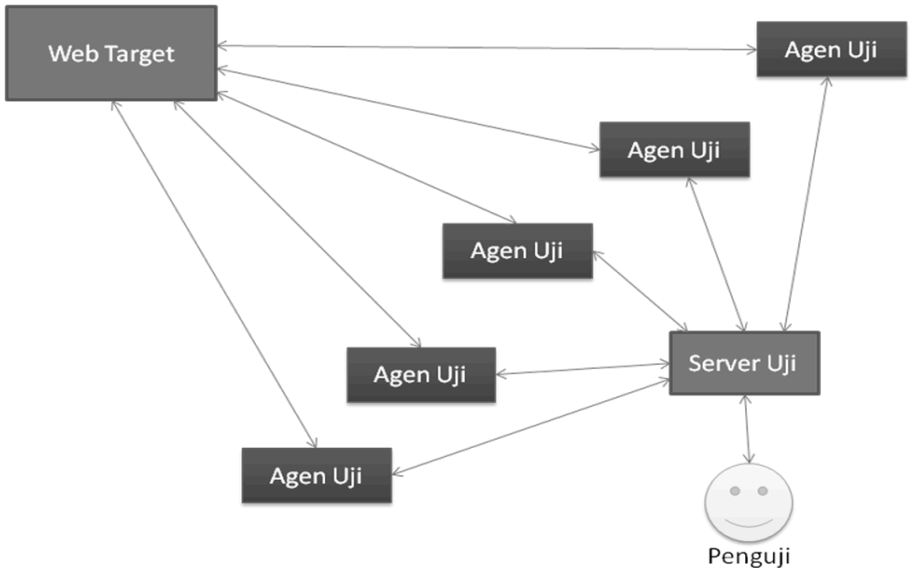
1. Pada bab 1 Pendahuluan dijelaskan tentang latar belakang, rumusan masalah, tujuan, hasil, metodologi dan sistematika pembahasan.
2. Bab 2 Kajian Pustaka, tentang 3 aplikasi sebelumnya dan perangkat lunak pembanding/ referensi.
3. Bab 3 Analisis Masalah dan Disain Solusi, berisi pembahasan masalah dan gambaran solusi serta disain dari solusi yang diajukan.
4. Bab 4 Implementasi dan Pengujian, membahas hasil implementasi dan pengujian perangkat lunak yang dilakukan.
5. Bab 5 Kesimpulan dan Potensi Pengembangan, berisi kesimpulan yang diambil berdasarkan hasil penelitian dan potensi pengembangan yang mungkin untuk penelitian selanjutnya.

BAB 2 STUDI PUSTAKA

Pada bagian ini dibahas tentang hasil studi pustaka yang berhubungan dengan peneltian ini. Diantaranya: aplikasi uji kapasitas dan performansi multi agen, kemudian aplikasi otomasi uji fungsional. Setelah itu juga dibahas tentang beberapa perangkat lunak yang dijadikan acuan dalam pengembangan perangkat lunak uji web enterprise terintegrasi berbasis FOSS [14], yaitu: Apace Jmeter, Solenium dan Watir. Perangkat lunak Apache Jmeter telah dibahas mendalam pada penelitian sebelumnya [4] sehingga di sini dijelaskan secara umum saja, sementara Watir pada versi terakhir mengacu pada Selenium 2. Oleh karena itu yang dibahas mendalam adalah Selenium, disamping menjadi acuan saat ini, juga memiliki arsitektur yang mirip dengan arsitektur perangkat lunak uji fungsional pada penelitian sebelumnya [4].

2.1. Model Dasar Otomasi Uji Multi Agen

Pada penelitian [2][3] dan [4] digunakan model otomasi uji multi agen seperti pada gambar 2.1.



Gambar 2.1. Model Otomasi Uji Multi Agen

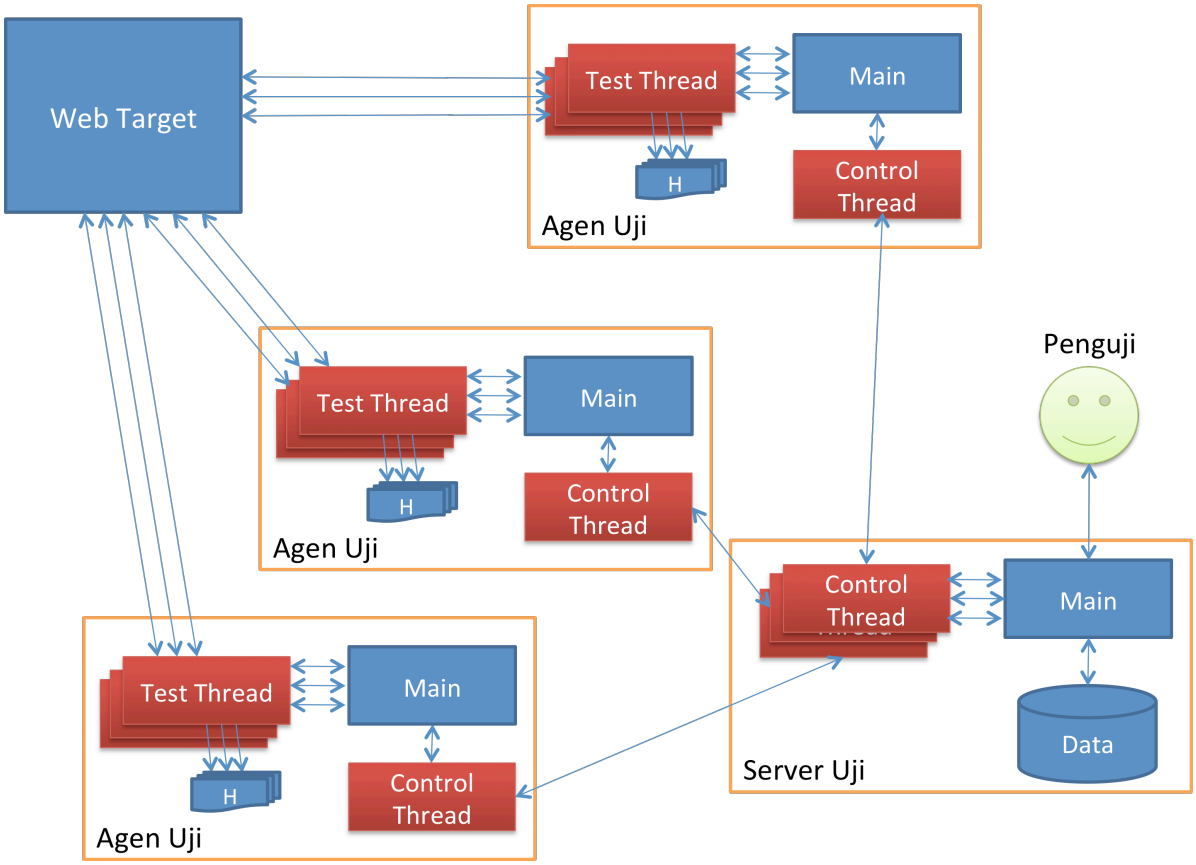
Pada model pada gambar 2.1. Penguji memasukkan perintah uji melalui Server Uji. Perintah mencakup instruksi uji (skenario) dan pesanan agar dieksekusi oleh Agen Uji tertentu (order). Agen Uji berperan sebagai *virtual user agent* yang menerjemahkan skenario ke dalam operasi web dalam bentuk browsing ke Web Target. Hubungan antara Agen Uji dengan Server Uji adalah *client-server* dimana Agen Uji sebagai *client* dan Server Uji sebagai

server. Konsekuensi dari model ini, bahwa hanya Server Uji yang harus memiliki alamat publik agar dapat diakses oleh Agen Uji, sementara Agen Uji tidak perlu memiliki alamat publik, melainkan cukup memiliki akses ke Server Uji dan Web Target. Dampaknya Agen Uji dapat di-*deploy* ke semua komputer yang terhubung ke internet (dapat mengakses Web Target). Ini artinya dapat mengerahkan Agen Uji dalam jumlah yang besar. Sebagai contoh, di Unpar, ada 1.300 an komputer yang tersebar di ruang kerja, kelas dan laboratorium. Semua komputer ini berpotensi dijadikan sebagai Agen Uji.

Penggunaan model pada gambar 2.1. lebih lanjut dijelaskan untuk uji kapasitas dan performansi (*load/ stress*) pada bagian 2.2 dan untuk uji fungsional pada bagian 2.3.

2.2. Model Uji Kapasitas dan Performansi Multi Agen

Pada penelitian [2] model pada gambar 2.1 di atas dikembangkan lebih lanjut dengan menggunakan multi threading seperti pada gambar 2.2.

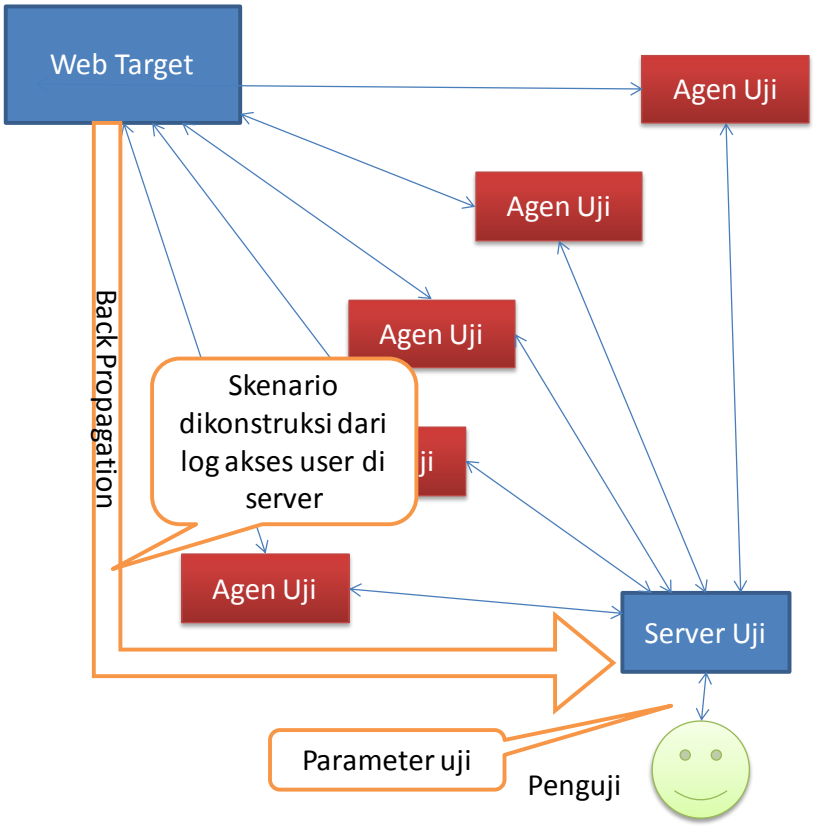


Gambar 2.2. Model Uji Kapasitas dan Performansi Multi Agen

Pada gambar 2.2. penguji mengirimkan perintah kepada Server Uji (SU). Perintah ini disimpan pada basis data. SU adalah aplikasi server yang listen pada port tertentu, misalnya

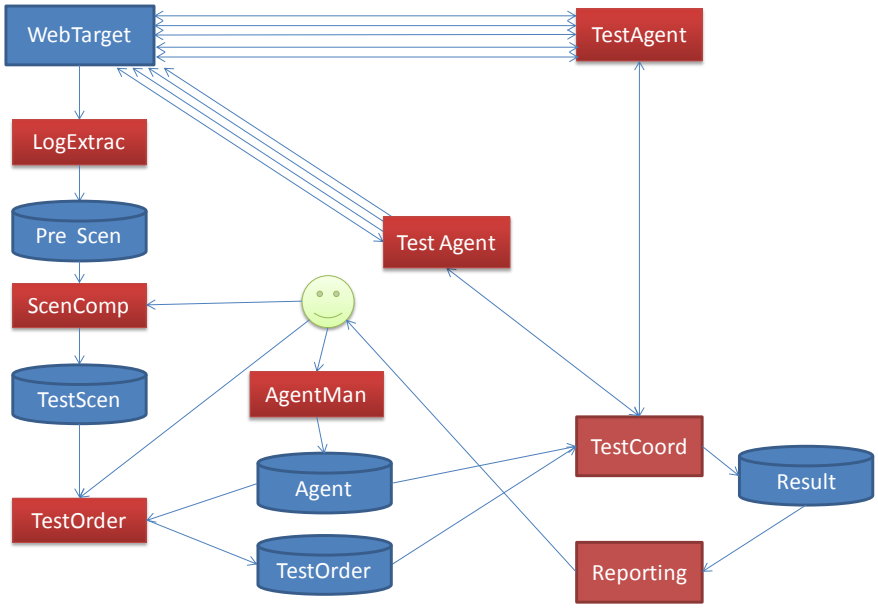
port 5000. Agent Uji (AU) menghubungi SU menggunakan Thread Control (TC). Untuk setiap TC di AU ditangani oleh 1 TC di SU, dengan demikian koneksi dapat bersifat *dedicated*. AU secara periodik mengirimkan pesan permintaan perintah kepada SU. SU memberikan perintah melalui TC berupa URL yang diuji, banyaknya UA yang harus dibangkitkan dan frekuensi request. Setelah AU menerima perintah maka akan dieksekusi dengan membangkitkan Test Thread (TT) sebanyak UA yang diminta. Dengan demikian 1 UA diemulasi dengan 1 TT. Misalnya untuk menghasilkan trafik request yang mengemulasi 1.000 UA dengan menggunakan 10 AU selama 1 dengan frekuensi 10x, SU akan memberikan perintah URL + 100 UA + 10x.

Pada penelitian [2] komunikasi antara Agen Uji dengan Server Uji terbatas pada instruksi berupa URL saja dengan *method* GET. Oleh karena itu, cukup menggunakan *text based* protocol dengan format yang <command> <data>. Pada penelitian [3] instruksi diperluas dengan mendukung *method* POST dan *cookie* sesuai dengan format pesan Hyper Text Transfer Protocol (HTTP) yang didefinisikan pada RFC 2616 [5], yang mendukung parameter berupa *header* dan *body*. Oleh karena itu protokol dikembangkan dengan menggunakan format JSON.



Gambar 2.3. Model Generator Skenario dengan Metode *Back Propagation*

Pada penelitian [3] juga diperluas dengan tambahan generator kasus uji berdasarkan jejak di server web menggunakan metode *back propagation*. Model yang digunakan dapat dilihat pada gambar 2.3. Model pada gambar 2.3 diimplementasikan dengan cara yang sama seperti pada gambar 2.2. Khusus generator menggunakan mod_security [6] pada server web Apache sehingga memungkinkan semua request dan respon tercatat detail dengan format yang memudahkan diproses [7]. Agar tester lebih mudah pada penelitian ini juga disediakan basis data untuk menyimpan agen, skenario, order dan hasilnya, yang diharapkan pada tahap berikutnya akan disediakan antarmuka berbasis web untuk mengelola data-data tersebut. Model lebih detail dapat dilihat pada gambar 2.4.

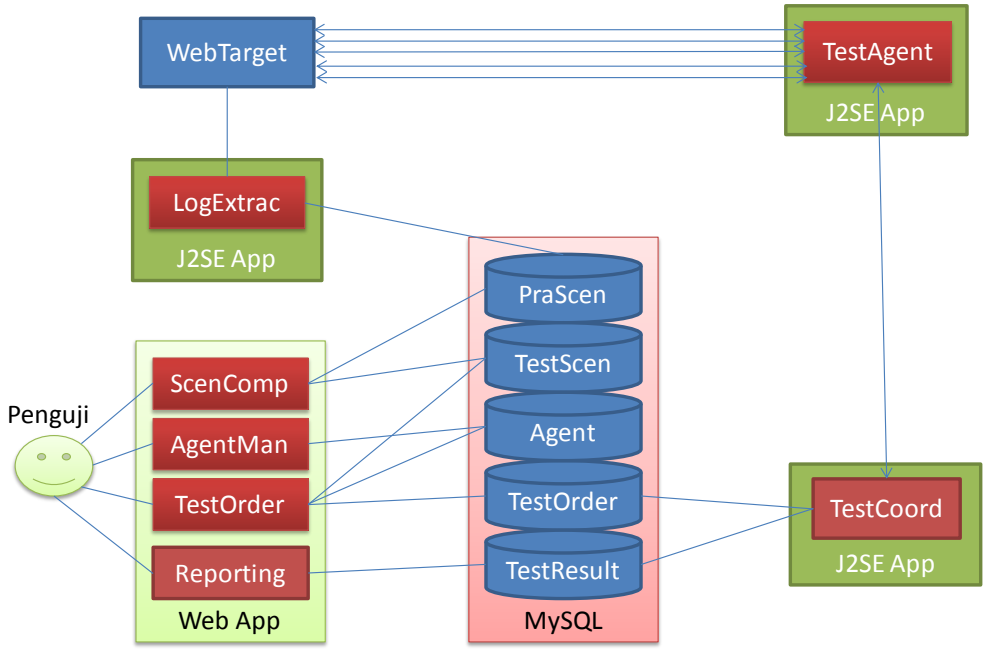


Gambar 2.4. Model Uji Kapasitas dan Performansi Multi Agen dengan Metode Back Propagation

Pada gambar 2.4, **LogExtrac** berfungsi untuk membaca log audit di web terget untuk menghasilkan data **PreScen**. Data **PreScen** merupakan daftar request yang dilakukan oleh user ke **WebTarget**. Data ini merupakan sumber untuk membuat skenario uji. **ScenComp** berfungsi untuk memfasilitasi penguji untuk mengkomposisi skenario uji berdasarkan data sumber **PreScen**. Dengan demikian konstruksi skenario benar-benar berdasarkan request user yang sesungguhnya. Dengan menggunakan aplikasi ini juga penguji dapat memasukkan varian dengan menentukan berapa varian yang diperlukan, key mana saja yang value-nya bervariasi. Semua skenario final disimpan ke data **TestScen** (skenario uji coba). **AgentMan** berfungsi untuk mengelola data TestAgent dan menyimpan datanya di basis data Agent. **TestOrder** (**TO**) berfungsi untuk membuat pesanan eksekusi uji coba ke **TestAgent**. Pesanan

dibuat berdasarkan skenario di TestScen dan data Agent. **TestCoord** berfungsi untuk berkomunikasi dengan semua **TestAgent**. Dalam komunikasi ini dilakukan proses otentifikasi dan pemberian perintah uji berdasarkan **TestOrder** dan menerima laporan hasil uji coba dari **TestAgent**. Semua hasil disimpan pada data **Result**. **TestAgent** berfungsi untuk melakukan uji beban berdasarkan perintah dari **TestCoord**. **Reporting** berfungsi untuk menampilkan laporan hasil uji berdasarkan hasil uji (Result) dan hal-hal yang perlu ditampilkan berkaitan dengan uji beban.

Model pada 2.4 diimplementasikan ke dalam aplikasi dengan arsitektur seperti pada gambar 2.5. LogExtrac, TestAgent dan TestCoord diimplementasikan pada lingkungan *Java 2 Standar Edition* (J2SE) [8] dengan teknologi *multi threading* [9]. Web App pada penelitian ini tidak diimplementasikan.

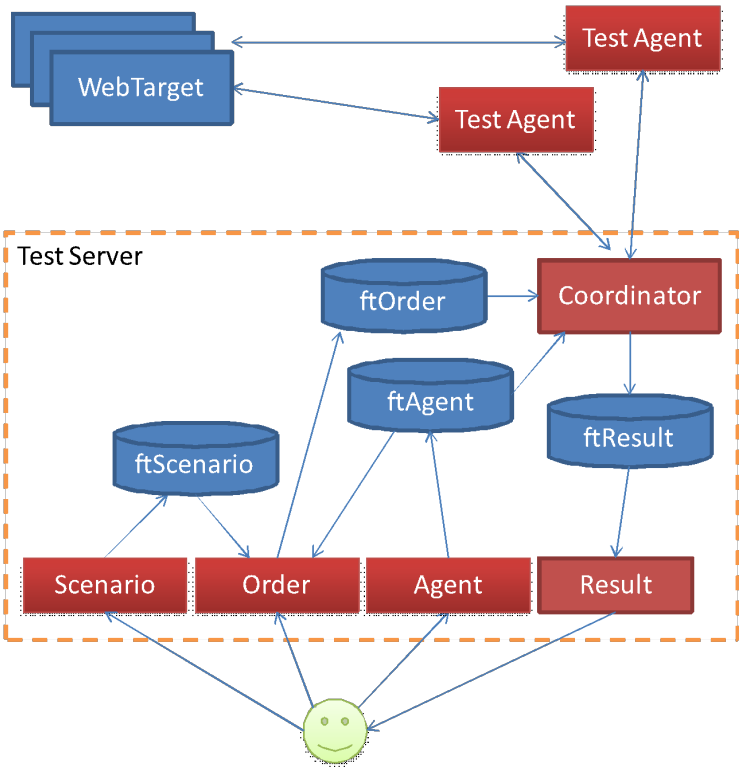


Gambar 2.5. Implementasi Model Uji Kapasitas dan Performansi Multi Agen dengan Metode Back Propagation

2.3. Model Otomasi Uji Fungsional Web Enterprise

Aplikasi otomasi uji fungsional web enterprise dikembangkan pada penelitian [4]. Model dasar yang digunakan sama dengan pada gambar 2.1. Model tersebut diperluas menjadi seperti pada gambar 2.6. Pada gambar 2.6 dapat dilihat bahwa, Test Server, terdiri atas 5

elemen, yaitu: Scenario merupakan aplikasi untuk menggenerate skenario. Skenario yang dihasilkan disimpan pada basis data ftScenario. Elemen kedua adalah Agent, merupakan aplikasi untuk mengelola Test Agent (TA) sehingga semua dapat dikoordinasikan dengan baik. Informasi yang dicatat adalah AgentName yang merupakan identitas dari setiap TA. Semua informasi tentang TA disimpan pada basis data ftAgent.



Gambar 2.6. Model Otomasi Uji Fungsional Web Enterprise

Elemen ketiga adalah Order, yang berfungsi untuk memesan agar suatu skenario dijalankan oleh sebuah TA. Semua pesanan disimpan pada basis data ftOrder. Elemen keempat adalah Coordinator, yang bertugas untuk mengkoordinasikan semua TA agar dapat bekerja sesuai dengan order yang ada di basis data Order. Setiap TA akan selalu berkomunikasi dengan Coordinator untuk menanyakan apakah ada order untuk dirinya, jika ada, maka akan memintanya kemudian akan menjalankan skenario yang ada di dalam pesanan. Semua hasil eksekusi order oleh TA diberikan kembali kepada Coordinator untuk disimpan di basis data ftResult. Elemen terakhir dari Test Server adalah Result yang akan menghasilkan laporan eksekusi dari semua skenario berdasarkan order yang diberikan kepada TA. Tester (penguji) dapat memasukkan skenario melalui Scenario, kemudian mengelola semua TA melalui Agent dan mengordernya melalui Order serta dapat memantau hasilnya untuk dianalisis melalui Result.

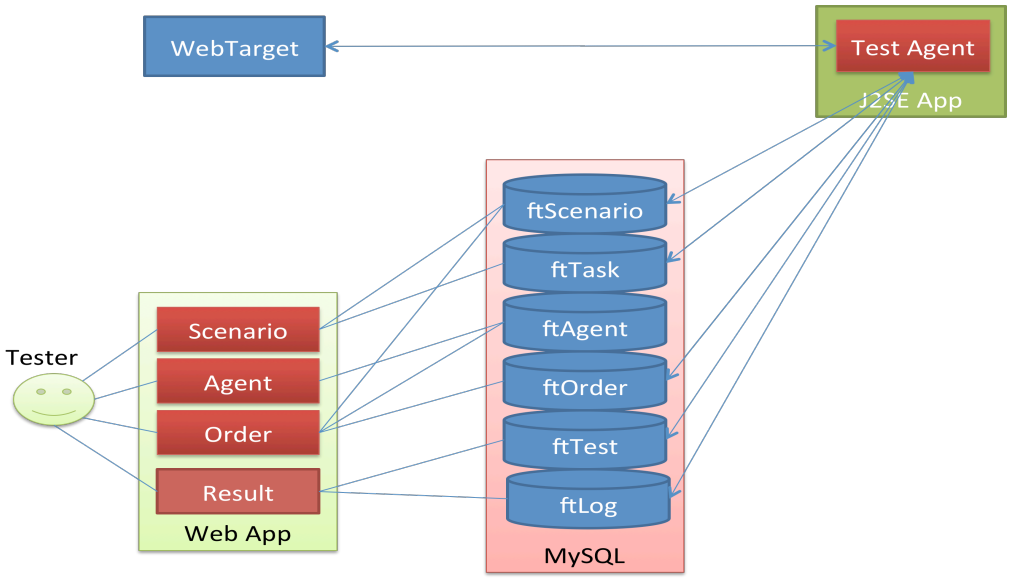
Agar Test Agent dapat menjalankan instruksi fungsional, baik pernyataan sederhana (*simple statement*), maupun *conditional* dan pengulangan (*repetition*), maka dibuat suatu bahasa dengan sintaks sebagai berikut:

<object> <action> [<param1> [<param2>]].

Di mana:

Object: elemen tampilan web.
Action: aksi dari setiap elemen.
Param1: parameter pertama yang menyatakan data atau variabel jika ada.
Param2: parameter kedua jika ada.

Model pada gambar 2.6 diimplementasikan dengan arsitektur seperti pada gambar 2.7. Fitur Scenario, Agent, Order dan Result diimplementasikan menjadi aplikasi berbasis web dengan menggunakan PHP. Basis data yang digunakan adalah MySQL. Sementara Test Agent diimplementasikan dengan J2SE dengan library HTMLUnit [10][11].



Gambar 2.6. Implementasi Model Otomasi Uji Fungsional Web Enterprise

2.4. Apache JMeter

Apache JMeter (selanjutnya disebut JMeter) [12][13] merupakan perangkat lunak *open source* berbasis Java desktop yang digunakan untuk menguji perilaku fungsional dan mengukur performansi suatu perangkat lunak berbasis web. Penjelasan lebih jauh dapat dilihat pada penelitian [4].

2.5. Selenium

Solenium [15] dikembangkan oleh Jason Huggins pada tahun 2004 dalam bentuk library Java Script. Solenium direlease dengan lisensi FOSS menggunakan kalusul lisensi Apache 2.0, sehingga dapat diunduh dan digunakan secara bebas dan gratis. Tahun 2006 sampai saat ini Selenium dikembangkan oleh Google untuk keperluan sendiri dan publik terutama Selenium 2 (Web Driver).

Saat ini, Selenium direlease dalam beberapa paket perangkat lunak yang memiliki fungsi tersendiri, yaitu:

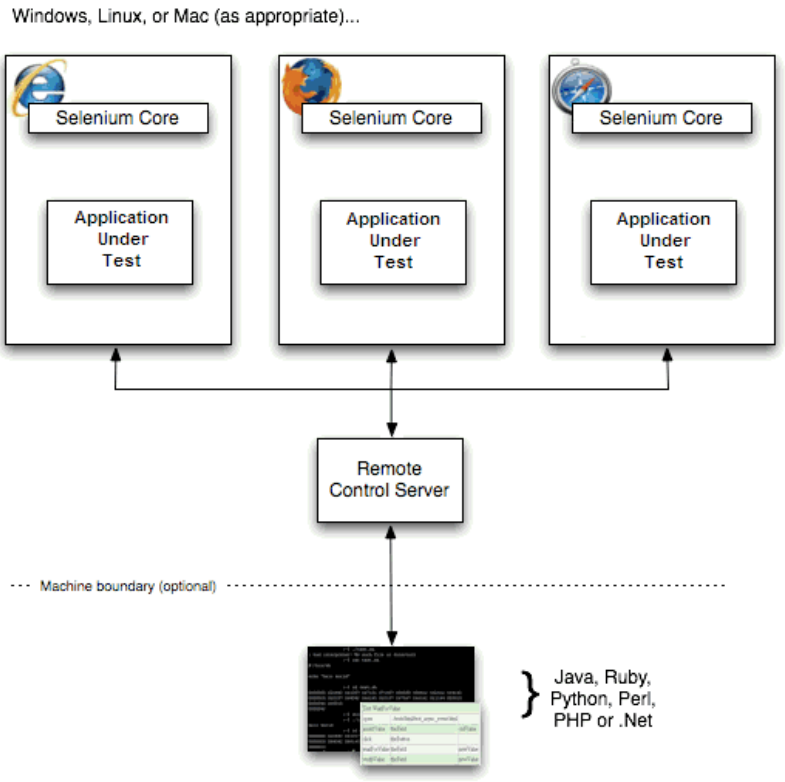
1. Selenium 1 [16]. Juga disebut sebagai Selenium RC (*Remote Control*). Merupakan paket utama yang berisi semua fungsi utama pengujian (*Selenium Core*). Paket ini dalam bentuk Java Script yang dikontrol oleh server babasis Java (Selenium RC Server).

2. Selenium 2 (*Web driver*) [17]. Selenium RC dikembangkan oleh Simon Stewart tahun 2006 menjadi Web Driver yang menyediakan API untuk mengakses browser secara langsung tanpa menggunakan Java Script. Pada awal tahun 2012, Simon Stewart dari Google (inventor of WebDriver) dan David Burns dari Mozilla bernegosiasi dengan W3C untuk menetapkan WebDriver sebagai *internet standard.*
3. Selenium IDE [18]. Merupakan prototipe untuk membuat *test script*. Selenium IDE adalah plugin pada browser Firefox yang menyediakan antarmuka yang mudah untuk membuat skenario test, dan juga menyediakan recording dari aksi pengguna yang dapat diekspor ke dalam berbagai bahasa. Akan tetapi Selenium IDE tidak memiliki kemampuan untuk menangani pengulangan dan kondisional.

4. Selenium Grid [19]. Memungkinkan untuk menggabungkan beberapa Selenium RC dalam rangka meningkatkan skalabilitas pengujian yang dijalankan pada beberapa lingkungan pengujian. Dengan demikian dimungkinkan dilakukan pengujian secara paralel dalam skala besar. Akan tetapi sayang dokumentasinya belum dibuat.

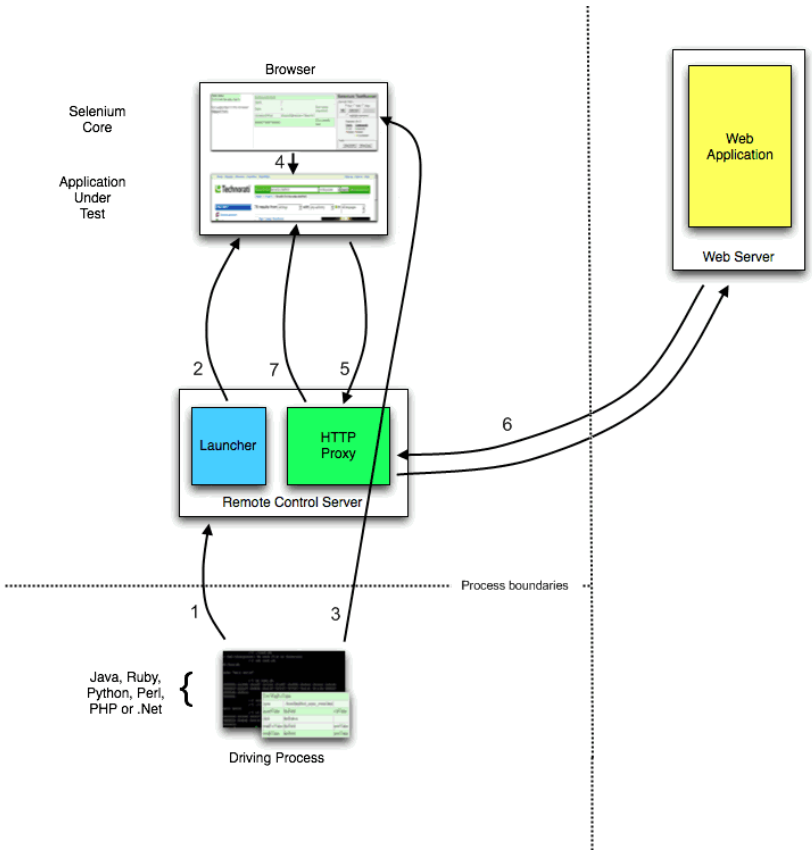
Arsitektur Selenium

Arsitektur Selenium dapat dilihat pada gambar 2.7. Pada gambar tersebut dapat dilihat bahwa Selenium RC menjalankan beberapa browser dan mengintepretasi perintah pengujian dalam bentuk *selenese command* dari skenario pengujian melalui browser tersebut dan me-record layaknya seperti HTTP Proxy antara browser dengan aplikasi yang diuji. Selenium core merupakan aplikasi Java Script yang bertugas menerjemahkan perintah *selenese command* kepada browser.



Gambar 2.7. Arsitektur Selenium RC

Gambara lebih detail mengenai cara kerja dari Selenium RC dapat dilihat pada gambar 2.8 dan 2.9. Pada gambar 2.8 Selenium RC menggunakan mode *Proxy Injection*, dimana semua perintah dilewatkan proxy dan dapat di-*record* oleh Selenium Server. Hal ini dilakukan untuk memenuhi persyaratan *Same Origin Policy* [20].



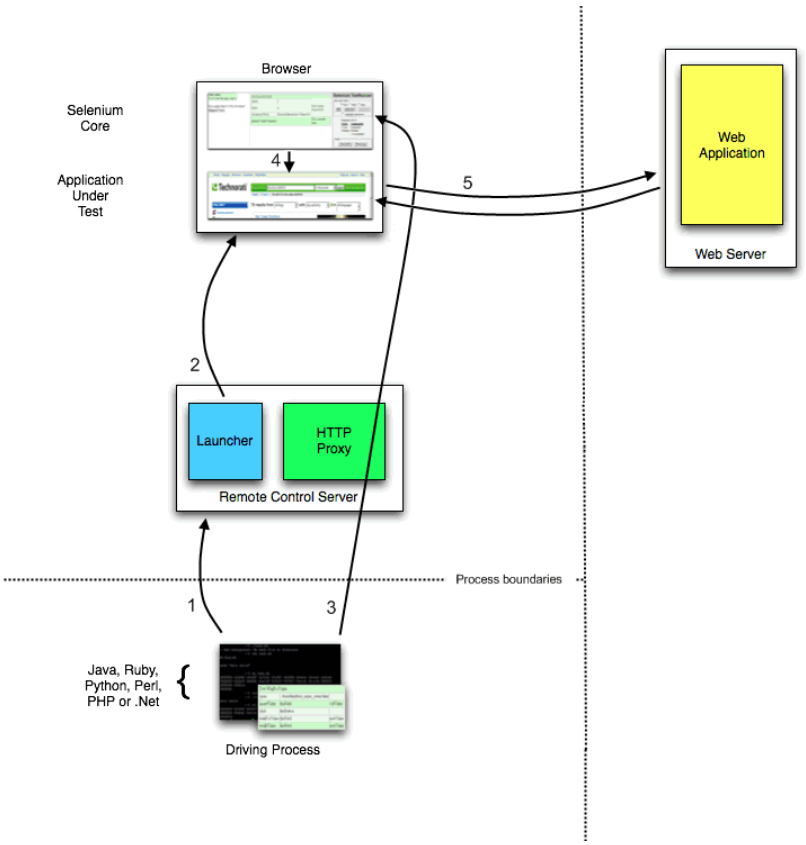
Gambar 2.8. Cara Kerja Selenium RC Dengan Mode Proxy Injection

Pada gambar 2.9 Selenium menggunakan mode *Heightened Privileges*, dimana tidak melaui proxy.

Pola Disain Testing Yang Didukung

Solenium mendukung beberapa pola disain testing [21], diantaranya:

1. *Static content*, untuk mengetahui apakah suatu situs berisi konten tertentu.
2. *Link,* apakah ada link atau tidak.
3. *Function,* dengan memasukkan suatu input, dan menghasilkan output tertentu.
4. *Dynamic element*, dicirikan dengan menggunakan identitas unik (ID) untuk mencari suatu elemen pada suatu halaman.
5. *Ajax,* menggunakan JS dan XML yang dinamis tanpa me-*reload* suatu halaman web.
6. Selain itu juga mendukung *data driven testing* dan *database validation.* Pada *data driven testing*, data yang digunakan untuk melakukan pengujian dapat diambil dari suatu basis data atau file eksternal (CSV atau text). Pola ini digunakan untuk membuat variasi data pada saat pengujian. *Database validation testing*, berfokus padan pencocokan antara hasil halaman web dengan isi dari suatu basis data. Pada pola ini, dimungkinkan untuk membandingkan hasil tampilan di web dengan hasil query ke basis data tertentu.



Gambar 2.9. Cara Kerja Selenium menggunakan mode *Heightened Privileges*

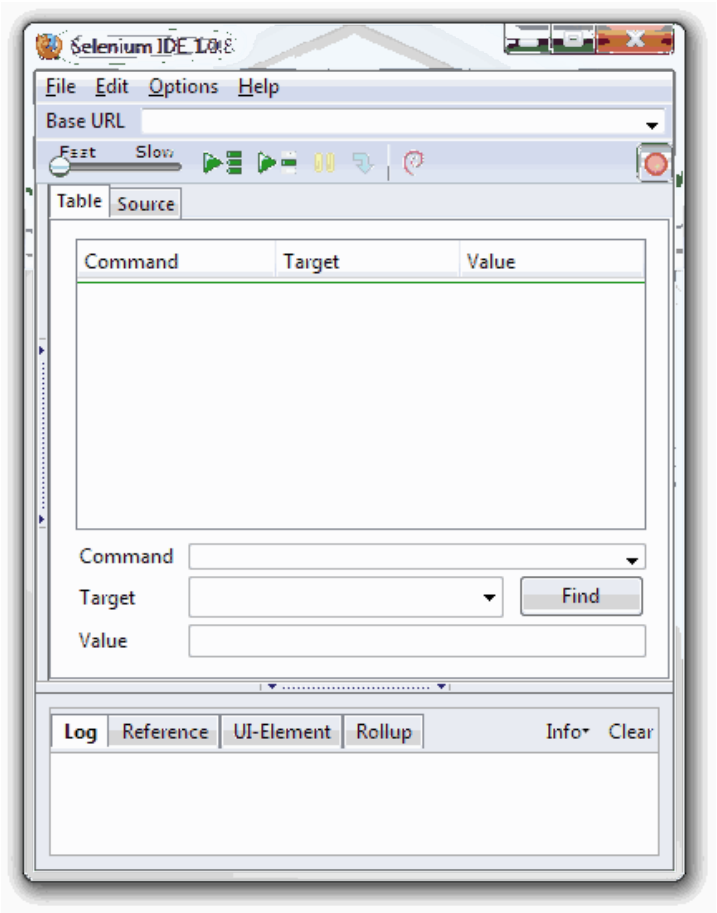
Solenium IDE

Solenium IDE merupakan perangkat lunak yang digunakan untuk membuat *test case* (skenario uji). IDE ini mendukung sintaks scrip selenium dan juga ada objek-objek elemen tampilan web. Pada gambar 2.10 dapat dilihat tampilan awal dari IDE ini.

2.6. Watir

Watir [22], dilavalkan “water”, adalah library keluarga Rubby yang *open-source* (BSD) untuk mengotomasi *web browser*. Watir juga memungkinkan untuk membuat skenario testing yang dijalankan di dalamnya. Watir sederhana dan fleksibel.

Watir mengendalikan browser seperti cara manusia melakukannya, dengan cara: meng-klik link, mengisi form dan menekan tombol yang ada di halaman web. Juga dapat melakukan pengecekan tampilan apakah mengandung informasi tertentu atau tidak.



Gambar 2.10. Selenium IDE

Whilst Watir hanya mendukung browser Internet Explorer pada Windows. Watir-WebDriver mendukung Chrome, Firefox, Internet Explorer, Opera dan HTMLUnit.

Watir telah digunakan untuk menguji situs-situs terkenal seperti: yahoo, SAP, Oracle, HP, facebook.

BAB 3 ANALISIS KEBUTUHAN DAN DISAIN SOLUSI

Pada bagian ini dijelaskan tentang analisis kebutuhan perangkat lunak uji terintegrasi web enterprise. Pada bagian ini juga dijelaskan bagaimana disain dan implementasi dari perangkat lunak tersebut.

3.1. Analisis Kebutuhan

Berdasarkan latar belakang, rumusan masalah dan tujuan dapat didefinisikan bahwa yang kebutuhan aplikasi lingkungan pengujian web enterprise terintegrasi mencakup:

1. **Pengelolaan Pengguna (*User Management*)**. Dalam hal ini pengguna dikelompokkan menjadi 2, yaitu: administrator dan tester. Administrator memiliki kewenangan untuk mengelola tester, selain semua wewenang yang dimiliki oleh tester. Tester dapat mengelola satu atau beberapa project. Dalam Pengelolaan Pengguna, selain dapat menambahkan dan mengedit, juga dapat mengubah/ menimpa password.

2. **Pengelolaan Proyek (*Project Management*).** Untuk setiap pengguna dapat mengelola beberapa proyek (*multi project*). Pengelolaan mencakup pembuatan, editing dan penghapusan. Untuk setiap proyek dapat diturunkan sejumlah testing.

3. **Pengelolaan Pengujian (*Testing Management*).** Pengujian dikategorikan menjadi 2, yaitu: uji fungsional (*functional*) dan uji performansi dan kapasitas (*load*). Pengelolaan mencakup pembuatan item uji, editing, penghapusan dan penurunannya ke dalam skenario dan turunannya.

4. **Pengelolaan Skenario dan *Task (Scenario and Task Management)***. Untuk setiap pengujian (*testing*) memiliki sejumlah skenario yang diturunkan menjadi task-task. Setiap task pada jenis uji fungsional merupakan instruksi dalam konstruksi bahasa dengan sintaks *<object> <action> [<param1> [<param2>]]*. Sedangkan untuk uji beban (*load*) dalam bentuk *request line* (URL), method, dan mungkin disertai dengan header dan parameter (*body*). Khusus untuk parameter, dimungkinkan ada varian value yang berfungsi untuk kastemisasi nilai setiap virtual user. Baik header maupun parameter memiliki format *<name> <value>*.

5. **Pengelolaan Agen (*Agent Management*)**. Pengelonaan ini diperlukan, agar setiap agen yang digunakan terdaftar dan statusnya termonitor. Agen juga ada 2 jenis: agen fungsional dan agen beban. Kedua jenis agen ini memiliki karakteristik yang berbeda,

sehingga dikelola pada menu yang berbeda, mengikuti jenis testingnya. Setiap tester memiliki kewenangan untuk mengelola agen. Agen sifatnya *shared*.

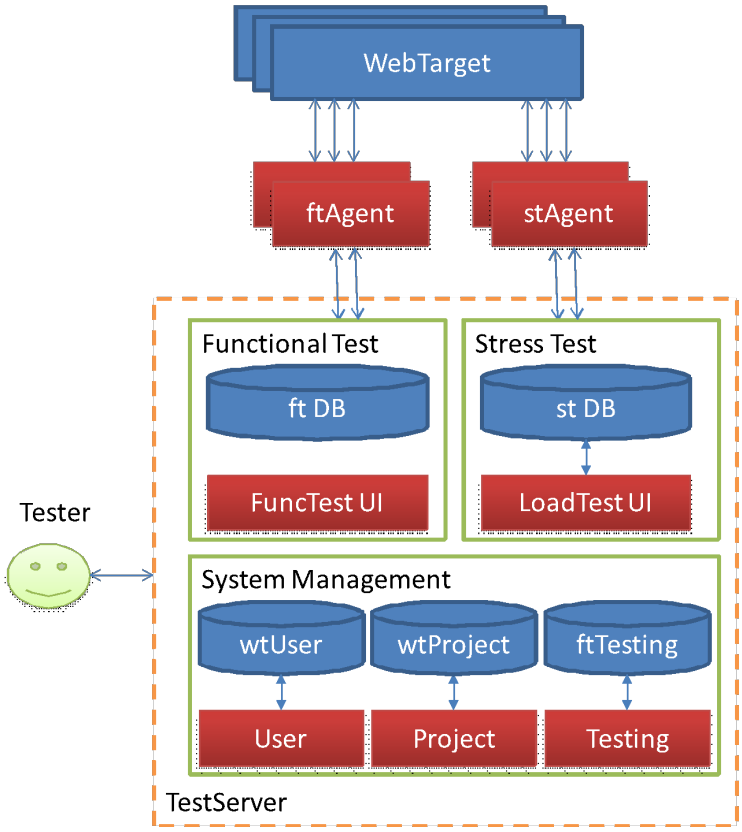
6. **Pengelolaan Pesanan Pengujian (*Order Management*).** Untuk mengeksekusi suatu skenario oleh suatu agen, terlebih dahulu dilakukan pemesanan. Pada kasus uji fungsional, pesanan hanya cukup memasangkan skenario dan agen, namun pada kasus uji beban, juga harus menyertakan parameter banyaknya *virtual user* yang didelegasikan pada suatu agen, berapa kali satu *virtual user* harus mengeksekusi secara simultan, dan berapa lama jeda waktu antar eksekusi (*think time*). Kewenangan akses order hanya diberikan kepada tester yang memiliki suatu skenario. Dengan demikian sifatnya eksklusif (tertutup).

7. **Pengelolaan Eksekusi dan Hasil (*Result Management*).** Untuk setiap order yang sudah disiapkan, status awalnya adalah *Idle*. Status agar dapat dieksekusi oleh agen harus diubah menjadi *Start*. Agen yang sedang mengeksekusi harus mengubah statusnya menjadi *InProgress.* Jika sudah selesai, maka agen akan mengubah statusnya menjadi *Finish.* Jika sudah berstatus *Finish,* maka hasilnya harus dapat dilihat, baik hasil global maupun hasil detail (log). Dalam hal ini untuk kasus uji fungsional dan load formatnya berbeda.

Selain kebutuhan fungsional di atas, batasan bahwa harus melakukan perubahan seminimal mungkin menyebabkan proses integrasi dilakukan tanpa mengubah struktur basis data kedua model (fungsional dan load), melainkan lebih untuk menambahkan hal-hal yang diperlukan. Untuk memenuhi kebutuhan di atas diajukan solusi dengan model seperti pada batian 3.1. Selanjutnya model tersebut dieksplorasi lebih jauh untuk menghasilkan disain global, detail dan diimplementasikan dengan teknologi web dengan PHP.

3.2. Model Solusi

Berdasarkan definisi kebutuhan pada 3.1 tersebut dapat diusulkan model solusi dengan arsitektur logika yang dapat dilihat pada gambar 3.1.



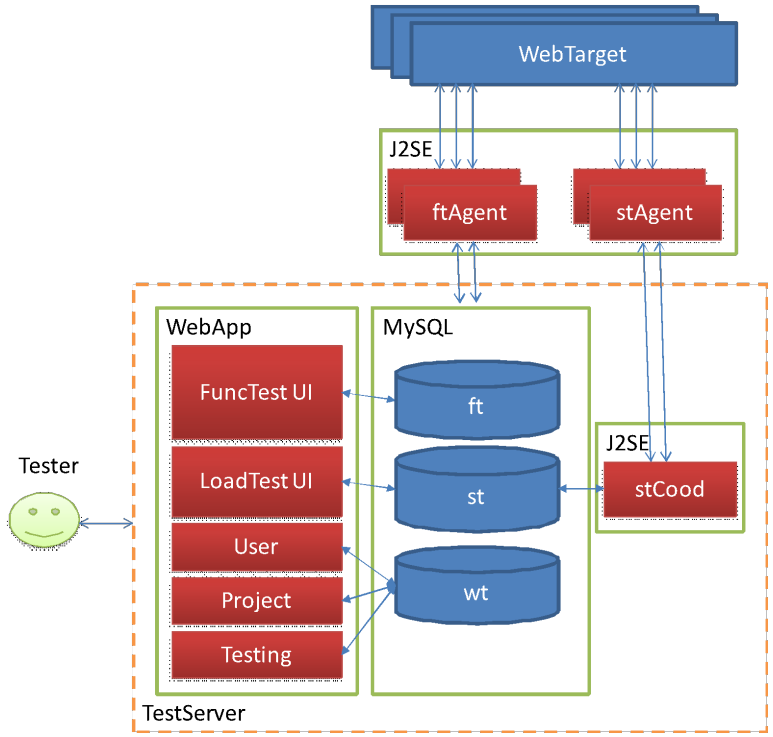
Gambar 3.1. Model Otomasi Uji Fungsional

WebTarget merupakan web server yang menjadi host dari situs web yang akan diuji. Pengujian dilakukan oleh 2 jenis agen, yaitu: ftAgent, merupakan agen untuk pengujian fungsional, dan stAgent untuk pengujian beban/ stress. Semua agen mendapat perintah uji berupa skenario dan task dari TestServer. TestServer menerima skenario dan task dari Tester (penguji). Skenario mewakili kasus uji fungsional dan beban, sedangkan task mewakili instruksi berupa langkah-langkah aksi yang dilakukan oleh user. TestServer teridi atas 3 bagian, yaitu: (1) *System Management*, yang mencakup fungsi pengelolaan user, proyek dan pengujian; (2) *Functional Test*, yang berisi semua komponen TestServer untuk uji fungsional (seperti pada gambar 2.6); dan (3) *Stress Test*, yang berisi semua komponen TestServer untuk uji beban (seperti pada gambar 2.4).

3.3. Disain Solusi

Model solusi pada bagian 3.2 diimplementasikan dengan teknologi sesuai dengan peruntukannya. Untuk agen, baik fungsional maupun beban diimplementasikan

menggunakan teknologi Java *multi threading* pada platform J2SE (*Java 2 Standar Edition*). Sedangkan TestServer diimplementasikan dengan teknologi web (Web App) menggunakan server Apache dan lingkungan pemrograman PHP. Untuk penyimpanan digunakan basis data MySQL. Lebih lanjut disain aplikasi dan teknologi yang digunakannya dapat dilihat pada gambar 3.2.



Gambar 3.2. Disain Solusi Lingkungan Uji Web Terintegrasi

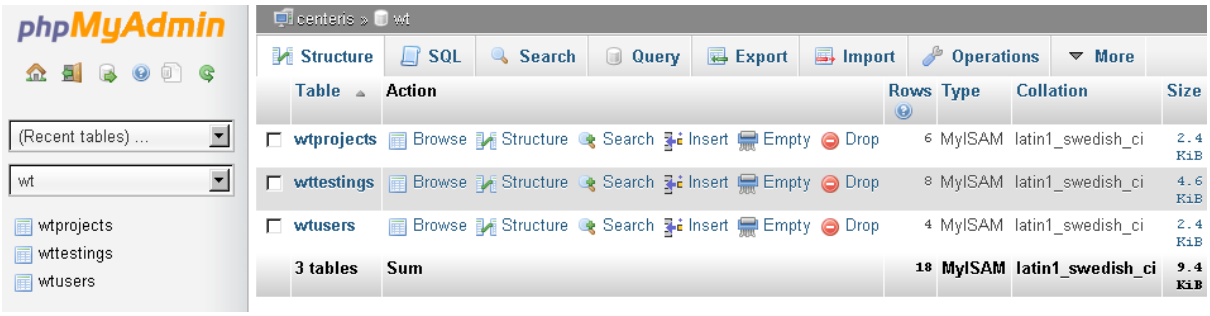
Pada gambar 3.2 dapat dilihat bahwa elemen Tester dapat mengakses pengelolaan User, Project, Testing dan semua antarmuka untuk uji fungsional (FuncTest UI) serta semua antarmuka untuk uji beban (LoadTest UI) pada satu aplikasi berbasis web (WebApp). Basis data terdiri atas 3, yaitu: wt untuk pengelolaan user, project dan testing, ft untuk semua data uji fungsional dan st untuk semua data uji beban. Pada sisi TestServer juga ada modul koordinator uji beban (stCoord) yang diimplementasikan pada lingkungan J2SE, untuk mengkoordinasikan agen-agen uji beban (stAgent). Sedangkan agen-agen uji fungsional langsung terkoneksi dengan basis data ft. Semua agen diimplementasikan dengan J2SE.

BAB 4 Hasil Implementasi dan Pengujian

Berikut adalah hasil implementasi basis data, program dan pengujian yang telah dilakukan. Pada bagian ini implementasi lebih difokuskan pada WebApp khususnya System Management dan LoadTest UI.

4.1. Implementasi Basis Data

Rangangan pada bab 3.2, telah diimplementasikan pada lingkungan MySQL versi 5.1, dan telah digunakan untuk melakukan uji coba dengan visualisasi seperti pada gambar 4.1 sampai 4.3.

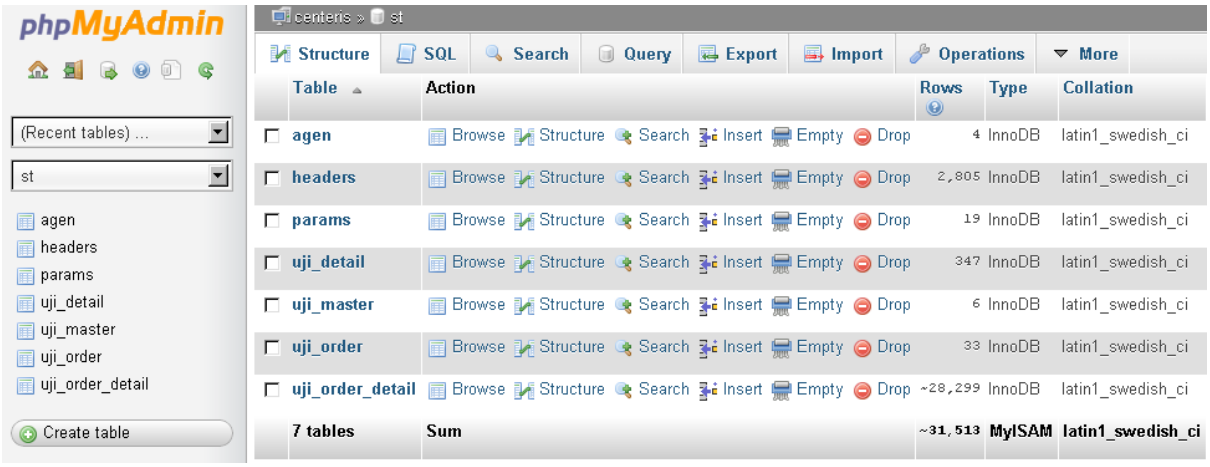


Gambar 4.1 Implementasi Basis Data System Management

Pada gambar 4.1. basis data **wt** untuk *system management* terdiri atas 3 tabel, yaitu: **wtusers** untuk menyimpan data tester, **wtprojects** untuk menyimpan data proyek dan **wttestings** untuk menyimpan data uji coba.

Gambar 4.2 Implementasi Basis Data Functional Test

Pada gambar 4.2. basis data **ft** untuk menyimpan data-data uji fungsional terdiri atas 9 tabel, yaitu: **ftobjec** dan **ftaction** untuk menyimpan bahasa fungsional, **ftscenario** dan **fttask** untuk menyimpan skenario uji dan turunannya. Sedangkan untuk menyimpan agen, order dan hasil eksekusi masing-masing pada tabel **ftagen**, **ftorder**, dan **fttest** serta **ftlog.** Untuk penyimpanan variabel selama eksekusi digunakan tabel **ftvar.**



Gambar 4.3 Implementasi Basis Data Stress Test

Dari gambar 4.3. basis data ft untuk menyimpanan data-data uji beban terdiri atas 7 tabel, yaitu: **agen** untuk menyimpana data agen. Skenario uji dan detailnya disimpan pada tabel **uji_master, uji_detail** serta **headers** dan **params.** Sedangkan untuk order dan hasil eksekusi disimpan pada tabel **uji_order** dan **uji_order_detail.**

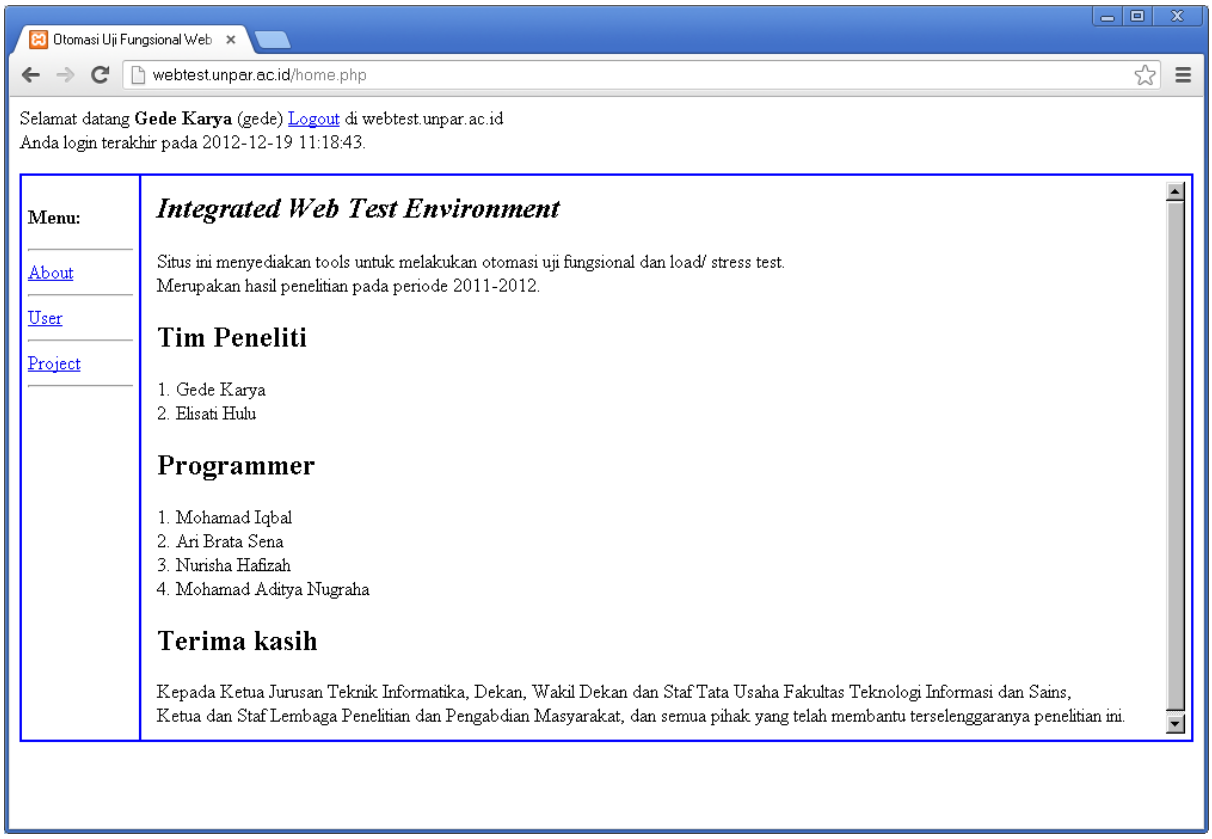
4.2. Implementasi Web App

Aplikasi Web (Web App) telah diimplementasikan, dan dapat diakses melalui situs http://webtest.unpar.ac.id. Beberapa halaman penting hasil implementasi dapat dilihat pada gambara 4.4 sampai dengan 4.10.

Halaman awal dari aplikasi uji terintegrasi ini, dapat dilihat pada gambar 4.4. Pada halaman ini user (admin/ tester) dapat melakukan login terlebih dahulu, sebelum diberikan halaman yang sesuai dengan wewenang aksesnya.

Gambar 4.4 Halaman Awal Uji Terintegrasi

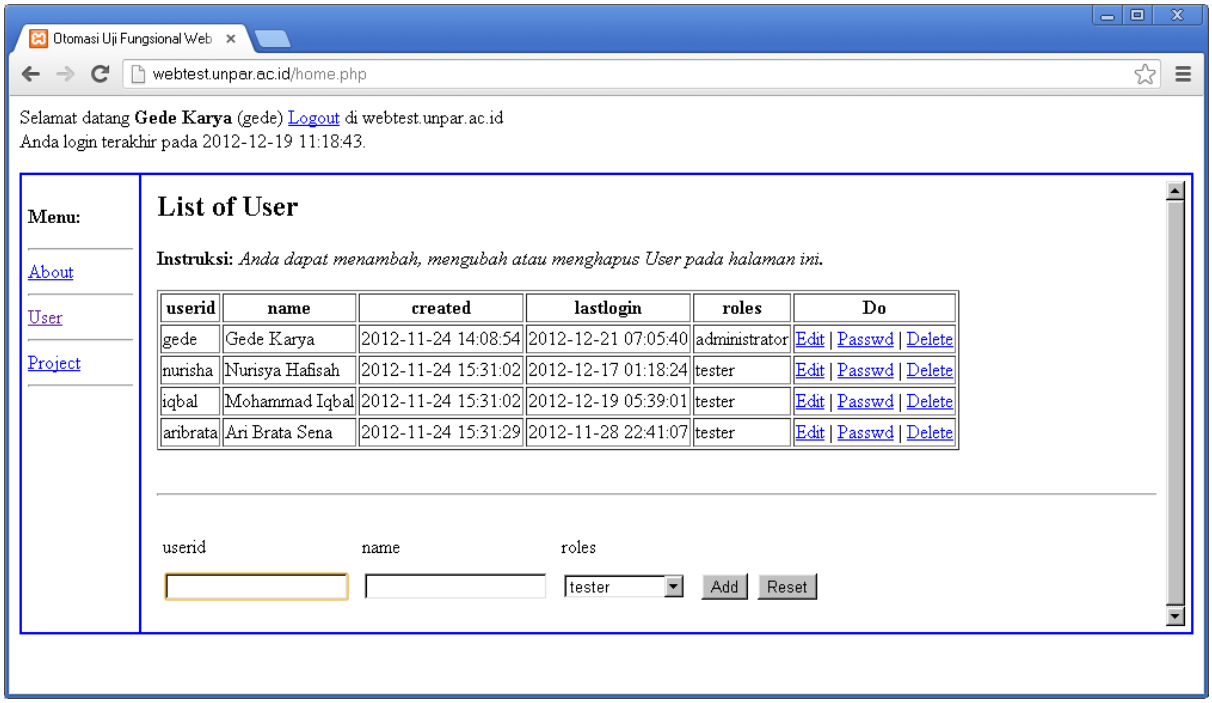
Setelah berhasil login, maka akan ditampilkan halaman web seperti pada gambar 4.5. Dalam contoh ini, yang login adalah gede (Gede Karya) dengan kewenangan sebagai administrator. Dengan demikian, fitur yang diberikan adalah pengelolaan user (User) dan proyek (Project).



Gambar 4.5 Halaman Awal untuk Administrator

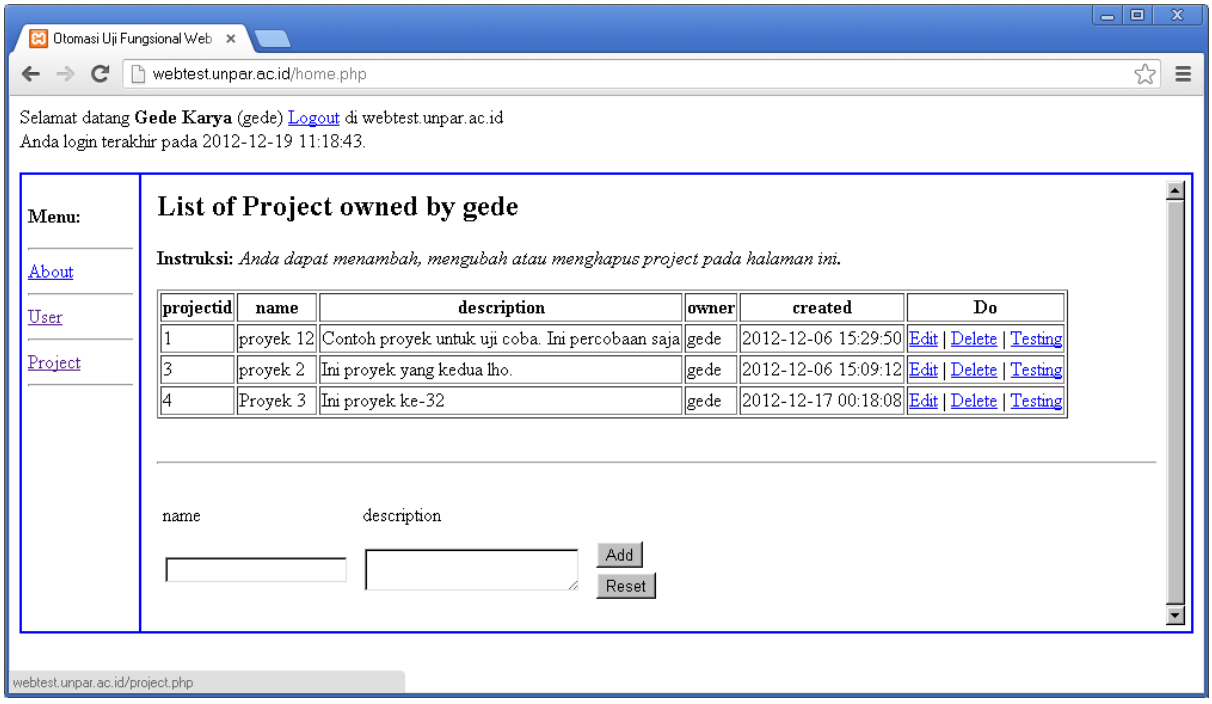
Jika user memiliki kewenangan sebagai tester saja, maka menu sebelah kiri hanya berisi About dan Project saja. Menu About akan menampilkan halaman tentang situs ini (seperti pada gambar 4.5).

Halaman untuk pengelolaan pengguna (User) dapat dilihat pada gambar 4.6. Di sini dapat dilakukan penambahan, editing, deleting data user dan pengubahan password.



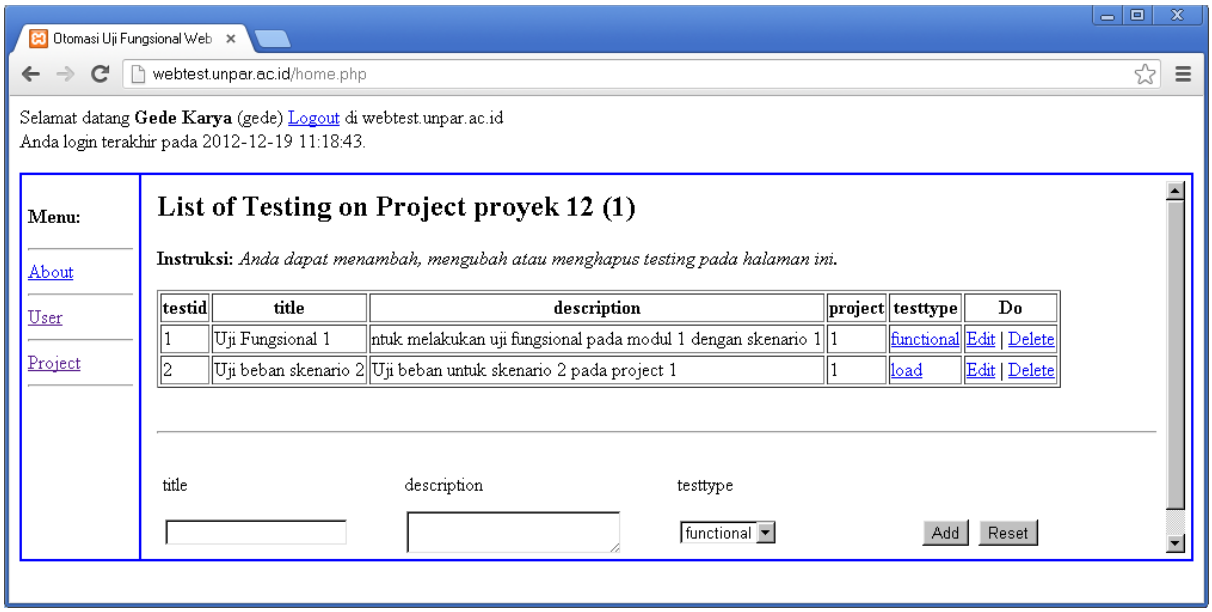
Gambar 4.6 Halaman Pengeloaan Pengguna (User)

Halaman untuk pengelolaan proyek (Project) dapat dilihat pada gambar 4.7. Di sini dapat dilakukan penambahan, editing, deleting data proyek dan navigasi ke halaman pengelolaan pengujian (testing).



Gambar 4.7 Halaman Pengelolaan Proyek (Project)

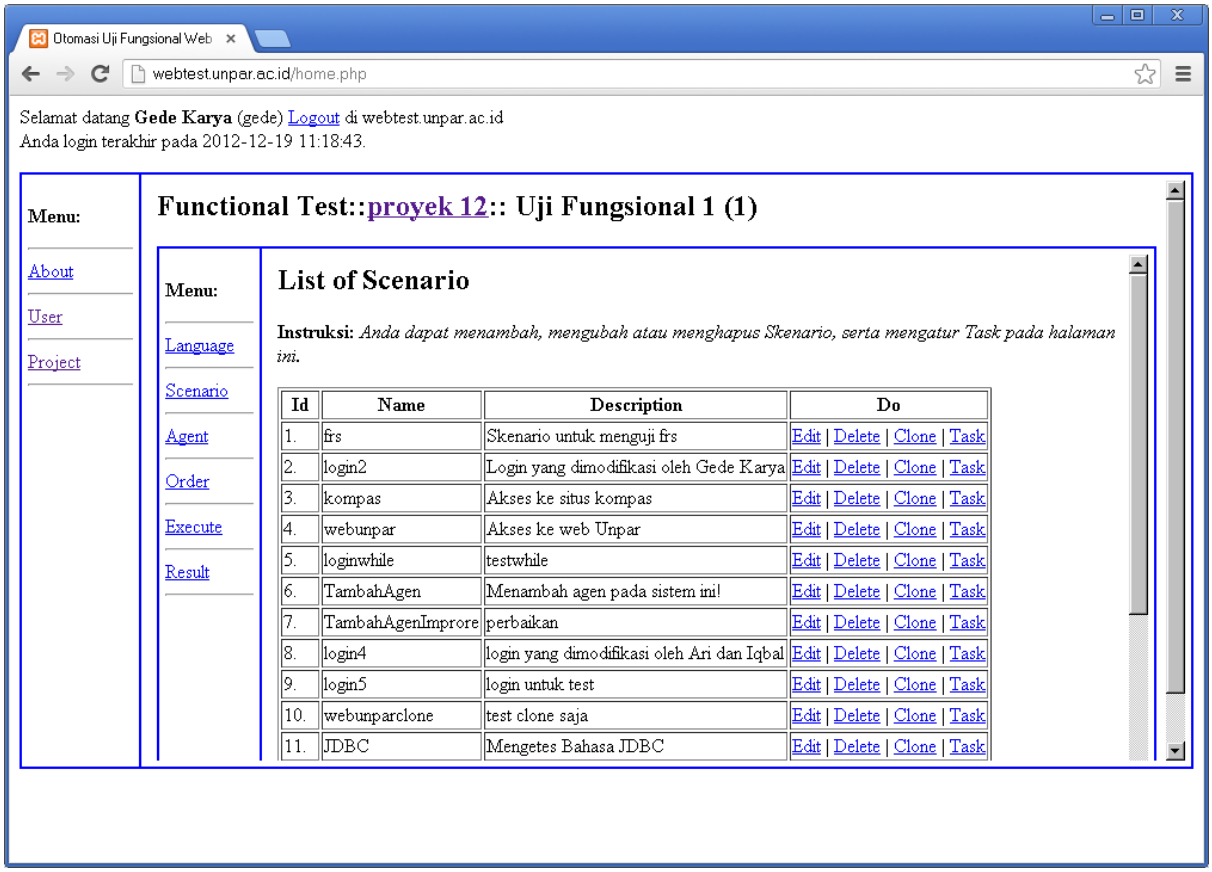
Jika kita pilih (klik) Testing, maka dapat mengelola semua pengujian pada proyek tersebut. Untuk lebih jelas dapat dilihat pada gambar 4.8.



Gambar 4.8 Halaman Pengelolaan Pengujian (Testing) Pada Suatu Proyek

Pada gambar 4.8 dapat dilihat, bahwa testing ada 2 jenis, yaitu: functional dan load. Jika kita pilih salah satu testing, maka akan muncul halaman sesuai dengan jenis testingnya. Pada

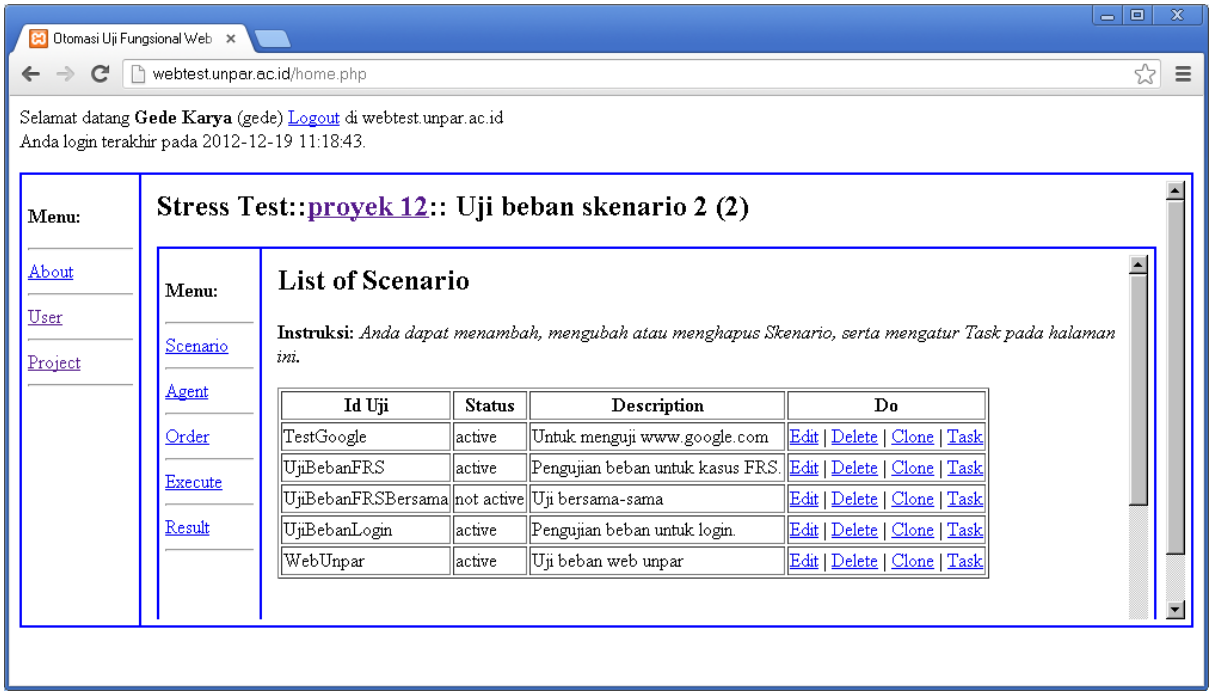
gambar 4.9 dapat dilihat testing fungsional, sedangkan pada gambar 4.10 testing beban (*stress/ load test*).



Gambar 4.9 Halaman Pengujian Fungsional (*Functional Test*)

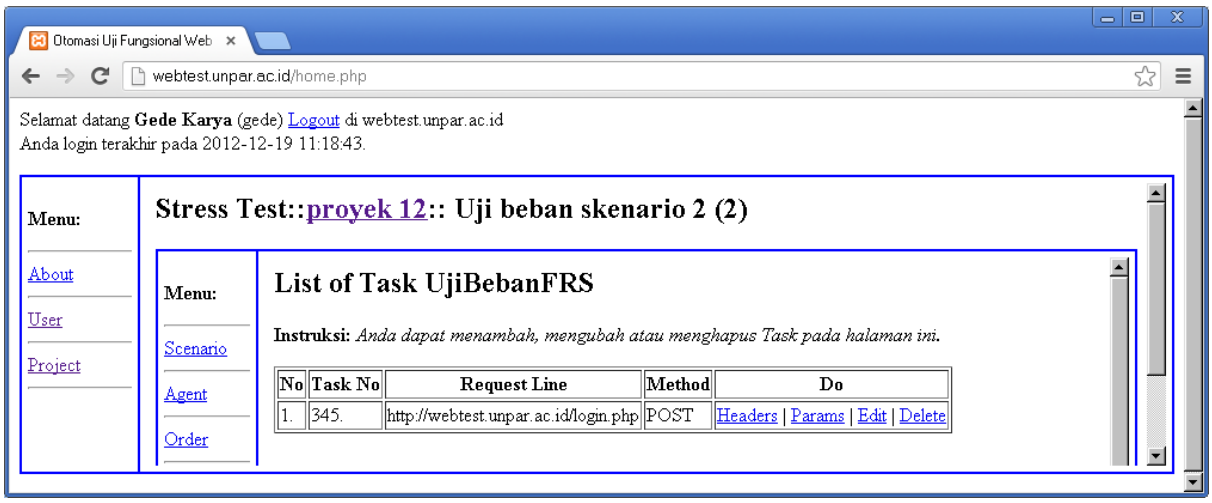
Pada gambar 4.9 dapat dilihat ada frame baru dengan menu-menu Language, Scenario, Agent, Order, Execute dan Result. Penjelasan lebih jauh tentang fungsi dari halaman-halaman ini dapat dilihat pada penelitian [3].

Selanjutnya kita lihat halaman implementasi uji beban pada gambar 4.10. Pada gambar 4.10, halaman uji beban (*Stress Test*) tampil dengan menu-menu: Scenario, Agent, Order, Execute dan Result.



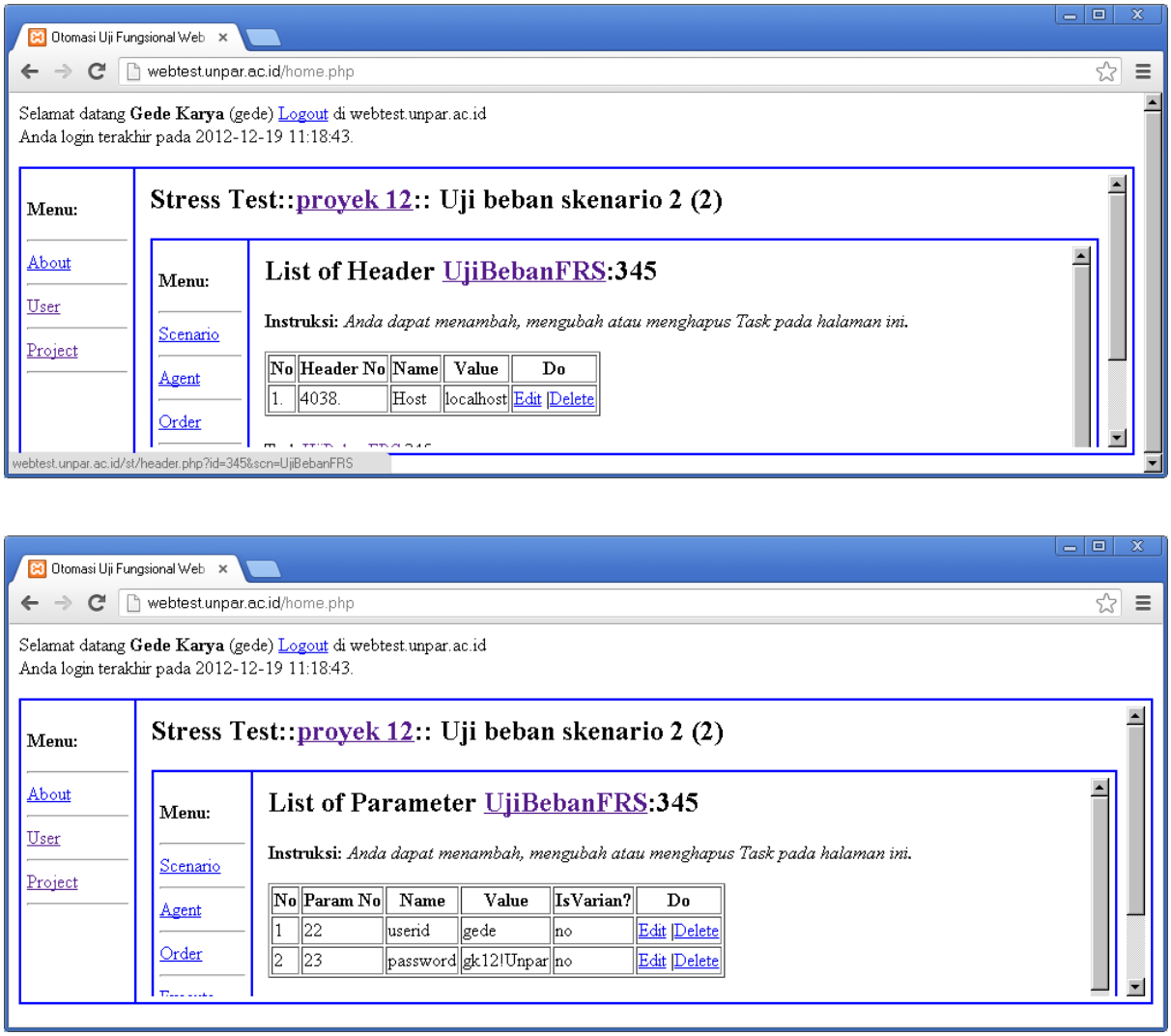
Gambar 4.10 Halaman Pengujian Beban (*Stress Test*)

Pada gambar 4.10 pada halaman utama ditampilkan skenario pada uji beban tertentu. Setiap skenario terdiri atas sejumlah task, seperti pada gambar 4.11.



Gambar 4.11 Halaman Task Uji Beban

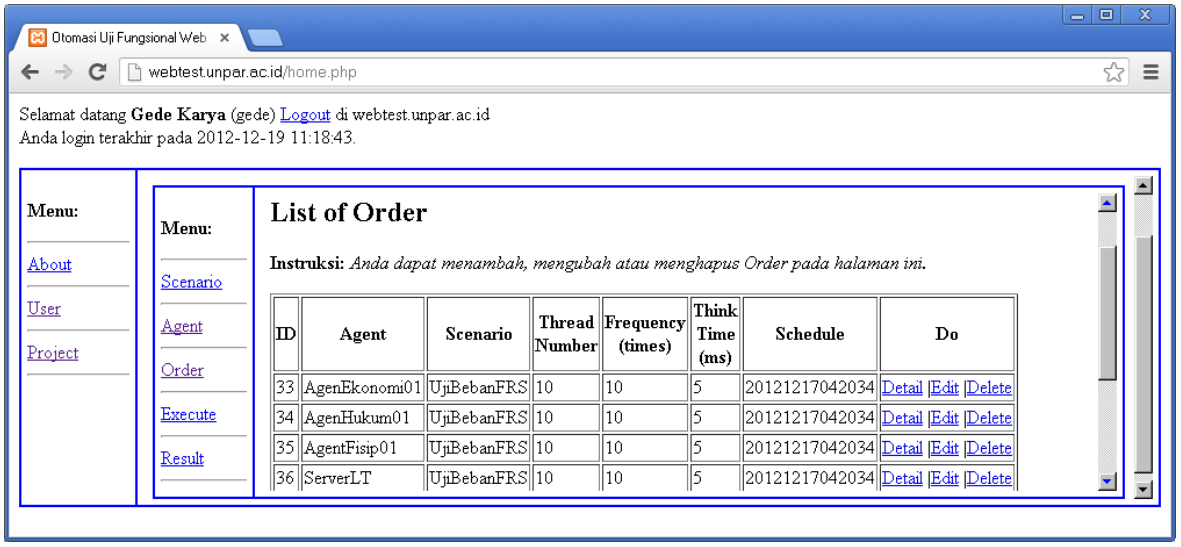
Pada gambar 4.11, setiap task terdiri atas Request Line (URL), Method dan dapat memiliki Headers dan Params. Contoh Headers dan Params dapat dilihat pada gambar 4.12.

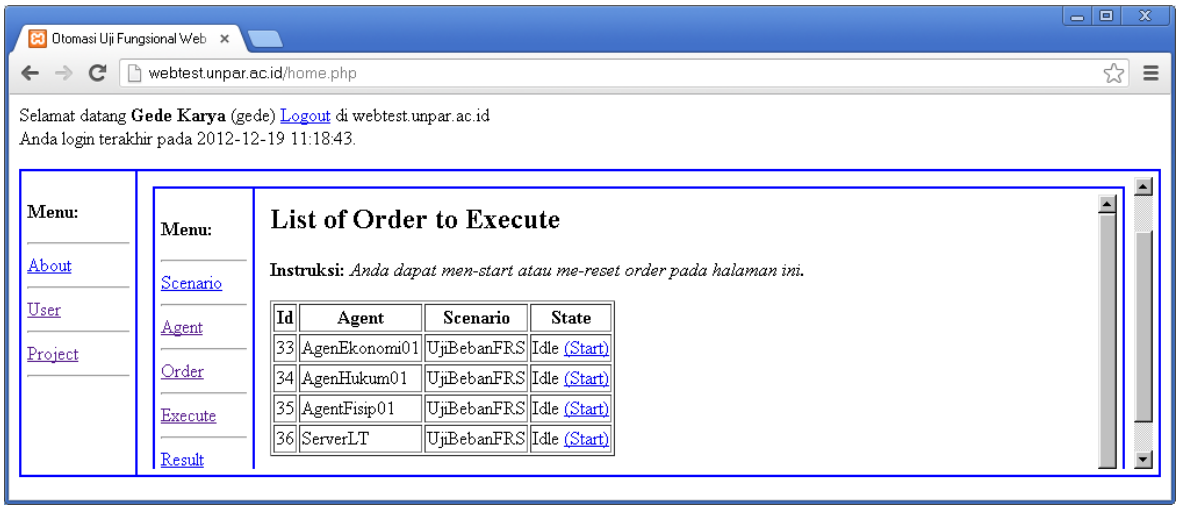


Gambar 4.12 Halaman Headers dan Params

Pada gambar 4.12, baik headers maupun param memiliki informasi name dan value. Khusus untuk params ada opsi IsVarian. Jika no, berarti sifatnya statis, akan tetapi jika yes artinya untuk setiap virtual user (VU) akan memiliki nilai sendiri.

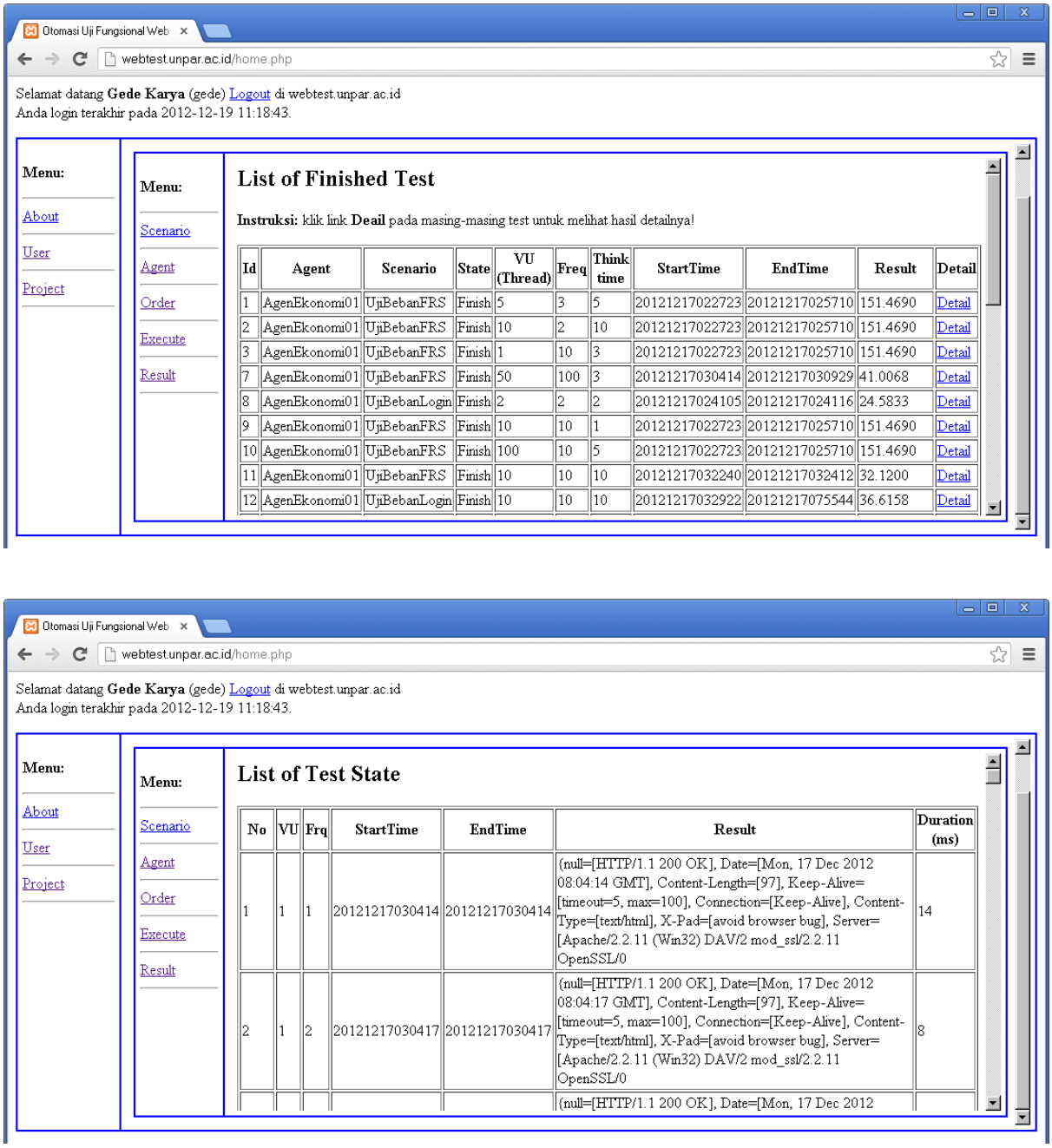
Selanjutnya untuk pengelolaan agen, order dan eksekusi dapat dilihat pada gambar 4.13. Pengeloaan agen, mencatat nama, password dan status setiap agen. Sedangkan untuk Order, selain mencatat agen dan skenario dieksekusi juga banyaknya virtual user (Thread) yang diemulasi, frekuensi dan waktu delay antar eksekusi (*think time*) serta jadwal. Pada Execute, semua order dapat dimonitor dan dikontrol untuk dijalankan (Start) atau di-reset.





Gambar 4.13 Halaman Agen, Order dan Eksekusi

Hasil eksekusi dapat dilihat setelah memilih menu Result seperti pada gambar 4.14. Pada gambar tersebut, hasil uji ditampilkan dalam bentuk global dan detail. Pada bentuk global, informasi result berupa waktu rata-rata eksekusi dalam *mili second* (ms). Sedangkan pada mode detail durasi yang digampilkan menunjukkan hasil eksekusi oleh setiap *virtual user* (VU) atas satu task.



Gambar 4.14 Halaman Hasil Uji (*Result*)

5.4. Pengujian

Fokus pengujian pada penelitian ini pada aspek fungsionalitas dari WebApp, khususnya untuk *System Management* (user, project, testing). Juga untuk fungsi-fungsi pengelolaan uji beban (*Stress Test*).

Untuk *Syste Managemen*, pengujian dilakukan menggunakan 4 user, yang terdiri atas 1 user administrator dan 3 user tester, dengan hasil seperti pada gambar 4.6. Untuk project masing-masing ada 2 project untuk 3 tester, yang juga dibagi masing-masing atas 2 testing untuk 4 project (lihat basis data wt pada gambar 4.1). Hasil dari pengujian menunjukkan fungsi-fungsi ini telah berjalan dengan baik.

Pada pengujian beban, dilakukan dengan menggunakan 4 agen (lihat gambar 4.13). Pengujian telah dilakukan menggunakan 32 Orde dengan 5 skenario (gambar 4.10). Hasilnya sebanyak 28 ribu record lebih (gambar 4.3), artinya telah terjadi 28 ribu lebih eksekusi *virtual user*. Dari pengujian ini, menunjukkan sistem sudah dapat mengelola pengujian beban dengan baik.

Dengan demikian semua implementasi sudah dilakukan dan berhasil dengan baik. Dari banyaknya pengujian juga dapat disimpulkan bahwa aplikasi sudah dapat berjalan dengan stabil.

BAB 5 KESIMPULAN DAN POTENSI PENGEMBANGAN

Pada bagian ini dijelaskan tentang kesimpulan yang diambil berdasarkan metodologi, kajian pustaka, dan hasil-hasil penelitian. Bagian ini diakhiri dengan potensi pengembangan yang dapat dilakukan untuk penelitian selanjutnya.

5.1. Kesimpulan

Berdasarkan hasil penelitian di atas, dapat disimpulkan:

1. Perangkat lunak uji yang dikembangkan sudah dapat berfungsi dengan baik, mencakup fitur: (a) pengelolaan sistem uji (*System Management*), (b) pengelolaan uji fungsional dan (c) Pengelolaan uji beban.
2. Berdasarkan hasil pengujian, bawa perangkat lunak sudah cukup stabil. Oleh karena itu sudah siap digunakan untuk melakukan uji sesuai dengan fungsinya.

5.2. Potensi Pengembangan

Dari hasil penelitian yang telah dilakukan, ada beberapa hal yang perlu dikembangkan yaitu:

1. Menggunakan perangkat lunak yang sudah dihasilkan pada penelitian ini untuk melakukan audit web enterprise.
2. Menggunakan perangkat lunak uji ini, khususunya uji beban untuk menguji berbagai konfigurasi dan membandingkan hasilnya.
3. Mengembangkan fitur statistik dan grafik hasil uji, dan dashboard fisual untuk memvisualisasi proses dan hasil pengujian.

DAFTAR REFERENSI

[1] Roger S. Presman, *Software Engineering A Practioner's Approach*, fitfh editon, McGraw-Hill, 2001

[2] Gede Karya, *Perangkat Lunak Uji Performansi Dan Kapasitas Situs Web Terotomasi Multi Agen, Laporan Penelitian*, Lembaga Penelitian dan Pengabdian Kepada Masyarakat (LPPM – Unpar), Juni 2011.

[3] Gede Karya, *Pembangkitan Skenario Dan Data Uji Performansi Dan Kapasitas Situs Web Terotomasi Multi Agen Dengan Metode Back Propagation, Laporan Penelitian*, Lembaga Penelitian dan Pengabdian Kepada Masyarakat (LPPM – Unpar), Desember 2011.

[4] Gede Karya, *Otomasi Uji Fungsional Web Enterprise, Laporan Penelitian*, Lembaga Penelitian dan Pengabdian Kepada Masyarakat (LPPM – Unpar), Agustus 2012.

[5] R. Fielding, J. Gettys, dkk. *RFC 2616 - Hypertext Transfer Protocol -- HTTP/1.1,* The Network Working Group of The Internet Society. 1999.

[6] *Mod Security*, http://www.modsecurity.org/, diakses: 6 Agustus 2011.

[7] *Mod Security Documentation > Audit Log > Data Format*, http://sourceforge.net/apps/mediawiki/mod-security/index.php?title=Data_Format#Audit_Log, diakses: 6 Agutus 2011.

[8] Bruce Eckel, *Thinking in Java*, Second Edtion, Printice-Hall, 2000.

[9] Chaffee, Alexander Day, *Building a Robust Multithreaded Server in Java*, Jguru Java Training, 1998

[10] *HTMLUnit*, http://htmlunit.sourceforge.net , diakses terakhir tanggal 21 Juli 2012.

[11] *HtmlUnit: A Quick Introduction*, http://java.dzone.com/articles/htmlunit-%E2%80%93-quick-introduction, diakses terakhir tanggal 22 Juli 2012.

[12] *Apache JMeter*, http://www.methodsandtools.com/tools/tools.php?jmeter, diakses terakhir tanggal: 12 Juni 2012.

[13] *Extending Jmeter*, http://www.jajakarta.org/jmeter/1.7/en/extending/index.html, diakses terakhir tanggal 13 Juni 2012.

[14] Web Testing, http://en.wikipedia.org/wiki/Web_testing, diakses tanggal 15 Juni 2012.

[15] Selenium Introduction, http://seleniumhq.org/docs/01_introducing_selenium.html, diakses tanggal 29 Agustus 2012

[16] Selenium 1 (RC), http://seleniumhq.org/docs/05_selenium_rc.html, diakses tanggal 29 Agustus 2012

[17] Selenium 2 (Web Drive), http://seleniumhq.org/docs/03_webdriver.html#chapter03-reference, diakses tanggal 29 Agustus 2012

[18] Selenium IDE, http://seleniumhq.org/docs/02_selenium_ide.html, diakses tanggal 29 Agustus 2012
[19] Selenium Grid, http://code.google.com/p/selenium/wiki/Grid2, diakses tanggal 29 Agustus 2012
[20] Some Origin Policy, http://www.w3.org/Security/wiki/Same_Origin_Policy, diakses tanggal 29 Agustus 2012
[21] Selenium Test Design Consideration, http://seleniumhq.org/docs/06_test_design_considerations.html, diakses tanggal 29 Agustus 2012
[22] Watir Home Page, http://watir.com, diakses tanggal 29 Agustus 2012