JITET (Jurnal Informatika dan Teknik Elektro Terapan)
Vol. 13 No. 3, pISSN: 2303-0577 eISSN: 2830-7062 http://dx.doi.org/10.23960/jitet.v13i3.6887

RANCANG BANGUN TEKNOLOGI OTOMATISASI SSH PADA *PROXY SERVER* MENGGUNAKAN *FRAMEWORK DJANGO* UNTUK SISTEM *MONITORING CACHE PROXY*

Muhammasd Nawwir Albi [1]

[1]Teknik Informatika, Universitas Lampung, Jl. Prof. Soemantri Brojonegoro, Bandar Lampung 35145

Keywords:
Cache, *Django*, Otomatisasi SSH, *Proxy Server*

Corespondent Email:
m.nawwiralbi130102@gmail.com

Abstrak. Saat ini perusahaan PT. Queen Network Nusantara, proses *monitoring* dilakukan secara manual dengan memeriksa server *proxy* satu per satu melalui terminal atau *Secure Shell* (SSH). Tujuan dari penelitian ini adalah untuk merancang dan membangun teknologi otomatisasi SSH pada server *proxy* untuk mendukung sistem *monitoring cache proxy* di PT. Queen Network dengan menggunakan *framework Django*. Penelitian ini menggunakan metode *Rapid Application Development* (RAD) dan pengujian fungsionalitas dan non fungsionalitas menggunakan *Blackbox*. Hasil pengujian menunjukkan fitur utama berhasil dikembangkan dengan 9 skenario dengan tingkat keberhasilan pengujian sebesar 88%, serta hasil pengujian *WebSocket* mampu mengumpulkan hingga 36.000 log per jam, dibandingkan dengan metode manual yang hanya mencatat sekitar 300 log per jam, menjadikannya 120 kali lebih cepat.

Abstract. *Currently, PT. Queen Network Nusantara conducts monitoring manually by checking proxy servers one by one through a terminal or Secure Shell (SSH). The purpose of this research is to design and build SSH automation technology on proxy servers to support the cache proxy monitoring system at PT. Queen Network using the Django framework. This research employs the Rapid Application Development (RAD) methodology, and functional and non-functional testing using the Blackbox method. The test results show that the main feature was successfully developed with 9 scenarios, achieving a testing success rate of 88%. Additionally, WebSocket testing showed the ability to collect up to 36,000 logs per hour, compared to the manual method which only recorded around 300 logs per hour, making it 120 times faster.*

1. PENDAHULUAN

PT. Queen Network Nusantara (QNN) merupakan perusahaan teknologi yang berfokus pada penyediaan layanan internet, atau yang lebih dikenal sebagai *Internet Service Provider* (ISP). Seiring dengan bertambahnya waktu, jumlah pengguna internet di perusahaan terus meningkat. Pertumbuhan ini berdampak pada kecepatan layanan internet yang ada di perusahaan. Oleh karena itu, sebagai upaya untuk meningkatkan kualitas layanan, perusahaan menggunakan perangkat lunak *Squid* yang dapat menjadikan sebuah server biasa menjadi *proxy server*. *Proxy server* memiliki kemampuan untuk menyimpan salinan data dari aset-aset pendukung halaman

website yang pernah diakses oleh pengguna internet ke dalam *proxy server*. Kemampuan tersebut dinamakan *cache proxy*. *Cache proxy* membantu mengurangi penggunaan *bandwidth* dan beban jaringan internet dengan menyimpan salinan data konten web yang telah diakses oleh pengguna di dalam local *cache* sehingga pengguna internet tidak perlu lagi mengunjugi server web asal untuk mengakses konten tersebut [1]. Hal ini memungkinkan pengguna untuk mengakses data dengan lebih cepat dan mengurangi permintaan langsung ke server asal, sehingga meningkatkan kecepatan dan efisiensi layanan internet yang disediakan oleh perusahaan.

PT. Queen Network Nusantara telah berhasil mengimplementasikan *cache proxy*, namun perusahaan masih memerlukan sistem pemantauan (*monitoring*) yang efektif. Sistem monitoring yang dimaksud adalah sistem yang bertugas untuk memantau *access log* dari pengguna internet yang terhubung dengan *proxy server* perushaan.

Saat ini proses monitoring *proxy server* masih dilakukan secara manual melalui proses penulisan perintah terminal secara langsung pada server *proxy*. Metode ini tidak hanya kompleks tetapi juga memakan waktu sekitar 5 menit untuk mendapatkan dan mencatat data *log cache proxy*. Masalah yang ditimbulkan dari metode ini adalah data yang di dapatkan kurang aktual. Untuk mengatasi kendala ini, PT. Queen Network Nusantara membutuhkan teknologi sistem monitoring *cache proxy* yang dapat mempermudah staf perusahaan dalam melakukan *monitoring* dengan mendapatkan data *access log* secara *real time* dan aktual.

Sistem monitoring *cache proxy* ini mebutuhkan teknologi otomatisasi SSH. Teknologi ini dapat menjalankan perintah terminal melalui *script* pada *proxy server*.

Untuk mewujudkan teknologi pada paragraf sebelumnya, perusahaan berencana mengembangkan teknologi tersebut menggunakan *framework Django*, karena *framework Django* sendiri medukung otomatisasi SSH melalui bahasa pemrograman *Python* dengan menggunakan paket *Paramiko*. *Paramiko* memiliki kemampuan untuk menjalankan *script* yang berisikan sebuah perintah konfigurasi pada *device*, kemudian perintah tersebut dikirimkan melalui protokol *Secure Shell* (SSH) yang akan menjalankan perintah tersebut [2]. Otomatisasi SSH pada *proxy server* berjalan dalam ruang lingkup *framework Django* dengan dukungan fitur *webscoket* oleh *library Django Channels* dimana dapat menjalankan *script Paramiko* secara berulang, sehingga proses pengambilan data *log cache proxy* bisa dilakukan secara *realtime* dan disimpan kedalam database. *Framework* tersebut menggunakan bahasa pemrograman *Python* [3]. Maka dari itu, berdasarkan topik diatas yang berfokus pada pengembangan teknologi Otomatisasi SSH pada *proxy server* dapat diangkat sebagai studi kasus penelitian yang berjudul "Rancang Bangun Otomatisasi SSH Pada *Proxy Server* Menggunakan *Framework Django* Untuk Sistem *Monitoring Cache Proxy*".

2. TINJAUAN PUSTAKA

2.1 *Proxy Server*

Proxy Server adalah sebuah sistem komputer yang berada di antara klien yang meminta dokumen web dan server target (sistem komputer lainnya) yang menyediakan dokumen tersebut. Dalam bentuk paling sederhana, *proxy server* memfasilitasi komunikasi antara klien dan server target tanpa mengubah permintaan atau balasan. Ketika kita mengajukan permintaan untuk suatu sumber daya dari server target, *proxy server* mengambil alih koneksi tersebut dan bertindak sebagai klien terhadap server target, meminta sumber daya atas nama kita. Jika balasan diterima, *proxy server* mengembalikannya kepada kita, sehingga memberikan kesan bahwa kita berkomunikasi langsung dengan server target [4].

Untuk memfasilitasi komunikasi antara klien dan server web, *proxy server* digunakan sebagai perantara komunikasi tersebut. Dalam beberapa kasus, *proxy server* dapat memodifikasi permintaan atau balasan, atau bahkan menyimpan balasan dari server target secara lokal untuk memenuhi permintaan yang sama dari klien yang sama atau klien lainnya di kemudian hari. Proses menyimpan balasan secara lokal untuk digunakan di masa mendatang dikenal sebagai *caching* [4].

2.2 *Cache Proxy*

Menambahkan infrastruktur baru di seluruh Internet bukanlah hal yang mudah. Solusi yang

lebih realistis adalah memaksimalkan pemanfaatan infrastruktur yang sudah ada. Salah satu pendekatan yang banyak digunakan untuk mengoptimalkan penggunaan sumber daya adalah dengan menerapkan *caching*, di mana konten dihasilkan sekali dan disimpan untuk digunakan kembali di masa mendatang. Ketika sebuah *request* yang sebelumnya telah diminta oleh *request* lain, konten tersebut dapat disajikan langsung dari *cache*. *Cache* ini dapat berada di berbagai lokasi di seluruh Internet [5].

2.3 Otomatisasi *Secure Shell* (SSH)

Otomatisasi SSH adalah proses menggunakan protokol *Secure Shell* (SSH) untuk menjalankan tugas secara otomatis pada sistem atau server jarak jauh tanpa interaksi manual yang berulang. Hal ini, ini melibatkan skrip atau alat yang memanfaatkan kemampuan SSH untuk mengelola koneksi terenkripsi, mengirim perintah, mentransfer file, atau menjalankan tugas lainnya secara efisien dan aman.. SSH sendiri merupakan teknologi yang memungkinkan pengguna untuk mengakses dan mengontrol server secara jarak jauh. SSH memiliki arsitektur *client/server*, di mana program server SSH, yang biasanya diinstal dan dijalankan oleh administrator sistem, menerima atau menolak koneksi yang masuk ke komputer *host*. Pengguna kemudian menjalankan program klien SSH di komputer lain untuk mengirimkan permintaan kepada server SSH. Semua komunikasi antara klien dan server terenkripsi dengan aman dan terlindungi dari modifikasi [6].

2.4 *Websocket*

WebSocket adalah teknologi canggih untuk membuat koneksi antara klien dan server (browser dan server) dan memungkinkan komunikasi antara mereka secara *real-time*. Perbedaan utama dengan *WebSocket* adalah memungkinkan Anda menerima data tanpa harus mengirim permintaan terpisah, seperti yang terjadi di HTTP. *Websocket* adalah protokol yang memungkinkan koneksi tetap terbuka tanpa terputus, sehingga memungkinkan transmisi data berlangsung dengan cepat. Keunggulan ini sangat penting ketika dibutuhkan pengiriman data dengan latensi rendah, sehingga data dapat segera diproses lebih lanjut [7].

Websocket, sebagai fitur baru dari HTML5, didefinisikan sebagai teknologi yang memungkinkan halaman web menggunakan protokol *Websocket* untuk komunikasi full-duplex dengan *host* jarak jauh. Teknologi ini memperkenalkan antarmuka *Websocket* dan mendefinisikan saluran komunikasi *full-duplex* yang beroperasi melalui satu *socket* di web. *Websocket* HTML5 secara efisien menyediakan koneksi socket ke internet dengan *overhead* yang minimal. Teknologi ini memberikan pengurangan besar dalam lalu lintas jaringan dan latensi dibandingkan dengan solusi Ajax *polling* dan *Comet*, yang sering digunakan untuk mentransmisikan data *real-time* guna mensimulasikan komunikasi *full-duplex* dengan mempertahankan dua koneksi HTTP. Oleh karena itu, *Web Socket* adalah teknologi ideal untuk membangun sistem komunikasi web waktu nyata yang skalabel [8].

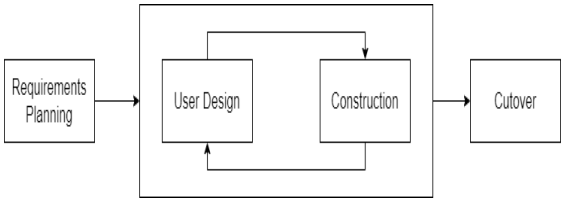
2.5 *Django*

Django menyediakan kerangka kerja tingkat tinggi yang memungkinkan untuk membangun aplikasi web dengan relatif sedikit baris kode. Kerangka kerja ini sederhana, kokoh, dan fleksibel, memungkinkan merancang solusi tanpa banyak hambatan. *Django* dibangun menggunakan *Python*, bahasa pengembangan aplikasi berorientasi objek yang menggabungkan kekuatan bahasa sistem seperti C/C++ dan Java dengan kemudahan dan pengembangan cepat dari bahasa skrip seperti *Ruby* dan *Visual Basic*. Ini memberikan penggunanya kemampuan untuk membuat aplikasi yang dapat menyelesaikan berbagai jenis masalah [9].

2.5 RAD (*Rapid Aplication Development*)

Rapid Application Development (RAD) merupakan metode pengembangan perangkat lunak yang menekankan siklus pengembangan yang cepat dan fleksibel. Model RAD digunakan untuk proyek yang membutuhkan pengembangan perangkat lunak yang memiliki kebutuhan yang berubah-ubah. Penggunaan *Rapid Aplication Development* (RAD) dalam penelitian ini memungkinkan keterlibatan langsung pengguna dalam proses pengembangan sistem. Jadi, pengguna dapat memberikan saran dan masukan secara langsung pada tahap pengembangan, sehingga

aplikasi yang dibuat dapat sesuai dengan kebutuhan dan harapan pengguna [10].



Gambar 2. 1 Diagram Metode Pengemmbangan RAD

Rapid Application Development (RAD) memiliki empat tahap yang terstruktur meliputi [10]:

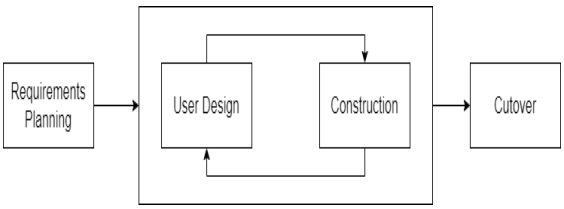
a. *Requirements Planning*: Pada tahap pertama ini, adalah perencanaan kebutuhan (*Requirements Planning*) dimana pengembang perangkat lunak berkonsultasi dengan pengguna untuk menentukan kebutuhan fungsional dan non-fungsional.
b. *User Design*: Pada tahap kedua ini, adalah desain pengguna (*User Design*) dimana pengembang perangkat lunak akan merancang desain sistem berdasarkan kebutuhan yang telah di tentukan pada tahapan sebelumnya.
c. *Construction*: Pada tahap ketiga ini, yaitu tahap konstruksi (*Construction*), pengembang mulai membangun fitur-fitur yang telah dirancang pada tahap sebelumnya. Setiap fitur akan diuji menggunakan skenario pengujian yang telah disiapkan. Jika ditemukan hasil yang tidak sesuai dengan harapan, proses iterasi akan dilakukan kembali pada tahap *user design* untuk memperbaiki dan menyempurnakan desain fitur tersebut.
d. *Cutover*: Pada tahap ini adalah tahap implementasi. Tahapan ini merupakan penggambungan dari seluruh fitur yang telah dibuat sehingga sistem dapat berjalan secara utuh sesuai yang telah direncanakan.

2.6 *Black Box Testing*

Black box testing berfokus pada persyaratan fungsional dari perangkat lunak. Dalam *black box testing*, penguji hanya mengetahui masukan yang diproses oleh sistem dan keluaran yang diharapkan, tanpa perlu mengetahui bagaimana sistem bekerja secara internal. Pengusjian ini dilakukan di seluruh siklus pengembangan perangkat lunak maupun siklus pengujian perangkat lunak, seperti dalam *regression testing*, *acceptance testing*, *unit testing*, *integration testing*, dan *system testing*. Jenis pengujian dalam teknik ini sepenuhnya berfokus pada pengujian fungsi dari aplikasi perangkat lunak [11].

Black Box Testing bertujuan untuk megujis apakah program tersebut berjalan sesuai dengan tugas yang telah ditetapkan tanpa melihat kode program yang dipakai [12].

3. METODE PENELITIAN



Gambar 3. 1 Diagram Metode Pengemmbangan RAD

Penelitian berjudul "Rancang Bangun Otomatisasi SSH Pada *Proxy Server* Menggunakan *Framework Django* Untuk Sistem *Monitoring Cache Proxy*" dilaksanakan selama tiga bulan menggunakan metode *Rapid Application Development* (RAD). Model RAD dipilih karena kemampuannya untuk mengakomodasi perubahan dan penyesuaian dengan cepat selama proses pengembangan.

3.1 *Requirements Planning*

Penelitian ini dimulai dengan tahapan pertama yang dikenal sebagai *requirements planning* atau perencanaan kebutuhan. Pada tahap ini, bertujuan untuk mengindentifikasi kebutuhan fungsional dan non-fungsional yang diperlukan untuk keberhasilan Otomatisasi SSH pada server *proxy*.

Sebelum mendapatkan hasil indentifikasi kebutuhan fungsional dan kebutuhan non fungsional, penelitian ini melakukan serangkaian kegiatan, seperti studi literatur, dan observasi.

Tahap Wawancara dilakukan dengan proses tanya jawab secara lisan kepada narasumber untuk mengumpulkan informasi langsung dari pengguna tentang kebutuhan dan masalah yang

dihadapi oleh pengguna terkait dengan *monitoring cache server*. Dibawah ini tabel 3.1 berisikan data daftar pertanyaan wawancara.

Tabel 3. 1 Daftar Pertanyaan Wawancara

Kode	Pertanyaan
P1	Seberapa penting menurut Anda *monitoring cache server* untuk kinerja keseluruhan sistem?
P2	Bagaimana selama ini perusahaan memantau performa *cache server*?
P3	Pernahkah ada mengalami masalah dengan performa *cache server* ? Jika ya, apa saja penyebabnya?
P4	*Framework* apakah yang digunakan untuk membangun keseluruhan sistem informasi ? dan mengapa ?
P5	Bagaimanakah proses *deployment* sistem informasi ini ?
P6	Apakah sistem informasi *monitoring cache server* ini memerlukan fitur *realtime* untuk memperbarui data *cache server* yang tersimpan di database ? jika ya, bagaimana cara implementasinya ?
P7	Apakah perusahaan memiliki standar pengkodean untuk aplikasi yang dikembangkan ?
P8	Apakah proses *query* pada *database* menggunakan sintaks SQL ?

Pada tahap observasi dilakukan pengamatan langsung terhadap sistem yang ada di PT. Queen Network Nusantara untuk melihat bagaimana sistem tersebut beroperasi. Objek observasi pada penelitian ini adalah melakukan perintah terminal server *proxy* secara langsung maupun melalui SSH dan proses penambahan data *log* pada tabel *database*

3.2 *Design*

Setelah kebutuhan awal teridentifikasi, langkah selanjutnya adalah melakukan Design. Design yang diperlukan antara lain *Entity Relationship Diagram* (ERD)

3.3 Construction

Setelah *design* sistem di buat, langkah selanjutnya adalah melakukan pengkodean dan pengujian teknologi Otomatisasi SSH dengan menggunakan *black box testing*. Di bawah ini tabel 3.2 berisikan sekenario pengujian teknologi otomatisasi SSH pada proxy server.

Tabel 3. 2 Tabel Sekanario Pengujian Otomatisasi SSH

Test ID	Skenario Pengujian	Hasil yang Diharapkan
TC01	Koneksi *WebSocket* berhasil	*WebSocket* terhubung dengan sukses.
		Pesan sukses diterima di klien (jika ada).
TC02	Koneksi *WebSocket* ditutup	Koneksi terputus tanpa *error*.
		Tugas periodik dibatalkan tanpa masalah.
TC03	Pengiriman pesan periodik berhasil	*WebSocket* mengirim pesan periodik berhasil.
TC04	Kesalahan pada periodik	Pesan error dikirim ke klien melalui *WebSocket*.
TC05	Data log diperbarui ke *database*	Data baru dari log *Squid* (*access.log*) disimpan ke *database*.
		Pesan *WebSocket*: data terakhir yang valid.
TC06	*Log file* tidak ditemukan	Pesan *error* dikirim ke klien.
		Tidak ada *crash* pada WebSocket atau server.

TC07	*WebSockes* tetap aktif saat koneksi klien terputus	WebSocket server menangani dengan benar tanpa *error*.
		Fungsi *disconnect*() dipanggil untuk membatalkan tugas periodik.
TC08	Data yang sudah ada tidak disimpan ulang ke database	Data yang sudah ada di database tidak disimpan ulang.
		Tidak ada data duplikat di *database*.
TC09	Koneksi *WebSocket* dalam mode beban tinggi	WebSocket tetap stabil tanpa *error*.
		Semua klien menerima pembaruan periodik sesuai interval.

4. HASIL DAN PEMBAHASAN

4.1 Hasil Wawancara

Adapun hasil wawancara, pada pada tabel 4.1 dibawah ini.

Tabel 4. 1 Hasil Wawancara

Kode	Jawaban
P1	sistem informasi *monitoring cache server* penting bagi perusahaan karena memiliki kemampuan untuk memberikan solusi bagi perusahaan untuk memecahkan permasalahan di atas dengan memberikan data-data yang aktual terkait data access log pengguna internet yang terhubung dengan proxy server perusahaan
P2	Selama ini perusahaan memantau *cache proxy* dengan menganalisis data yang diperoleh dari *SSH* yang terhubung ke server *proxy*.
P3	Pernah terjadi kegagalan fungsi *proxy server squid* diakibatkan *overload* pengguna
P4	Sistem ini menggunakan *Framework Django*.
P5	*Production* sistem informasi ini berjalan di server perusahaan.
P6	Ya, sistem informasi ini membutuhkan fitur *realtime* dalam proses memperbarui data, dikarenakan perusahaan membutuhkan data aktual terkait *access log cache proxy* sebagai bahan analisis untuk menentukan masalah di masa depan.
P7	Ya perusahaan memiliki standar pengkodean agar mempermudah proses pemeliharaan kedepanya
P8	Tidak, Proses *query database* yang dilakukan hanya perlu menggunakan model yang telah di sediakan *django* saja

4.2 Kebutuhan Fungsional

Adapun kebutuhan fungsional, dijabarkan pada tabel 4.2 dibawah ini.

Tabel 4. 2 Kebutuhan Fungsional

Kode	Kebutuhan Fungsional
KF1	Melakukan koneksi otomatis ke server *proxy* menggunakan SSH untuk mengambil data *access log* pengguna internet yang terhubung dengan *proxy server* perusahaan dengan fitur *websocket* yang di miliki oleh *framework Django* secara *real time*.

4.3 Kebutuhan Non Fungsional

Adapun kebutuhan fungsional, dijabarkan pada tabel 4.3 dibawah ini.

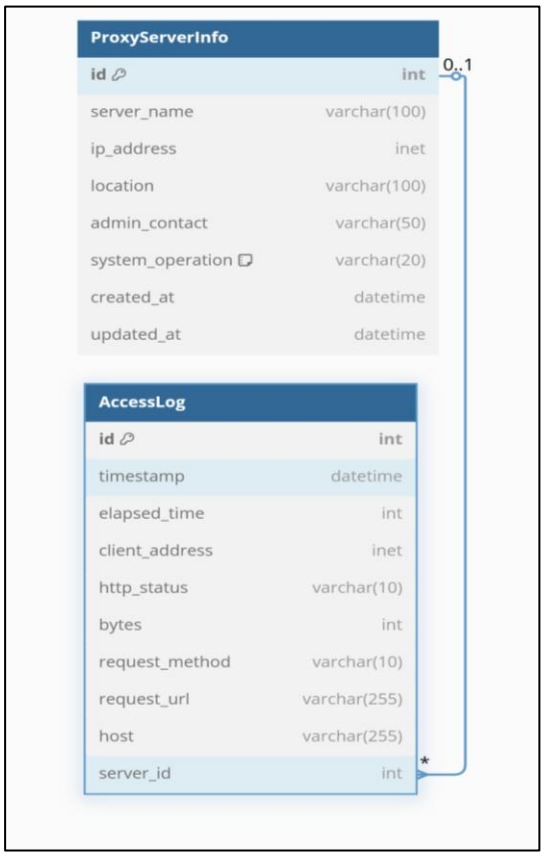
Tabel 4. 3 Kebutuhan Non Fungsional

Kode	Kebutuhan Non Fungsional
KNF2	Otomatisasi SSH harus diimplementasikan dengan *django websocket* agar proses *SSH* bisa dilakukan secara *realtime*
KNF3	Penulisan kode harus menggunakan standar

	pengkodean yang konsisten, agar dapat memberikan kemudahan dalam proses pemeliharaan.
KNF6	Membuat model *Django* yang mempresentasikan diagram ERD (Entity Relation Diagram) atau tabel database pada sistem

4.4 *Entity Relationship Diagram* (ERD)

Adapun *Entity Relationship Diagram* (ERD) berisikan informasi terkait tabel *database* sistem, diperlihatkan pada gambar 4.1 dibawah ini.



Gambar 4. 1 Tabel Database Sistem

Diagram ERD di atas menggambarkan struktur tabel dan relasi data untuk sistem monitoring *cache proxy*. Tabel utama adalah *ProxyServerInfo*, yang menyimpan informasi dasar tentang server proxy seperti nama server, alamat IP (Internet Protocol), lokasi, kontak administrator, dan sistem operasi yang digunakan. Tabel ini menjadi pusat referensi bagi tabel *Accesslog*. Sedangkan, untuk tabel *Accesslog* berisikan informasi lengkap untuk riwayat pengguna internet yang terhubung ke *proxy server*. Adapun dibawah ini tabel 4.4 memberikan penjelasan untuk setiap kolom pada tabel *Accesslog*.

Tabel 4. 4 Deksripsi Kolom Pada Tabel Accsess Log

Tabel *Accesslog*	
Nama Kolom	Deskripsi
timestamp	Waktu pencatatan *log* akses dalam format *string*, dengan panjang maksimum 50 karakter.
elapsed_time	Waktu yang dihabiskan untuk permintaan dalam milidetik, menunjukkan seberapa cepat permintaan diproses.
client_address	Alamat IP klien yang melakukan permintaan melalui *proxy Squid*.
http_status	Status HTTP yang dikembalikan dari permintaan (misalnya 200, 404), panjang maksimum 50 karakter.
bytes	Jumlah byte yang dikirim ke klien sebagai respons dari permintaan yang dilakukan.
request_method	Metode HTTP yang digunakan dalam permintaan (misalnya GET, POST), panjang maksimum 200 karakter.
request_url	URL lengkap dari permintaan yang diajukan klien.
host	Nama *host* tujuan dari permintaan, opsional (boleh kosong).
server	Referensi ke tabel *ProxyServerInfo*, yang berisi informasi tentang server *Squid* yang mencatat log ini.

4.5 Pengkodean *Model Database Django*

Adapun hasil pengkodean *model database django* yang mempresentasikan tabel-tabel *database server*, diperlihatkan pada gambar 4.2 dibawah ini.

```python
class ProxyServerInfo(models.Model):
    """
    Tabel ini berisi informasi tentang server proxy.
    """

    server_name = models.CharField(max_length=50)
    ip_address = models.GenericIPAddressField()
    username = models.CharField(max_length=50)
    password = models.CharField(max_length=50)
    location = models.CharField(max_length=50)
    admin_contact = models.EmailField()
    system_operation = models.CharField(max_length=50)
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)

class AccessLog(models.Model):
    """
    Tabel ini berisi log akses dari Squid Proxy.
    """

    timestamp = models.CharField(max_length=50)
    elapsed_time = models.IntegerField(
        help_text="Waktu yang dihabiskan untuk permintaan"
    )
    client_address = models.GenericIPAddressField()
    http_status = models.CharField(max_length=50)
    bytes = models.IntegerField(help_text="Jumlah byte dikirim ke kli
    request_method = models.CharField(max_length=200)
    request_url = models.URLField()
    host = models.CharField(max_length=50, blank=True, null=True)
    server = models.ForeignKey(ProxyServerInfo, on_delete=models.CASC
```

Gambar 4. 2 Hasil Pengkodean Model Database Django

Model *ProxyServerInfo* menyimpan informasi mengenai *server proxy* yang mencakup berbagai atribut penting. *server_name* berisi nama unik yang digunakan untuk mengidentifikasi *server proxy*. *ip_address* menyimpan alamat IP dari server tersebut yang digunakan untuk koneksi jaringan. *username* dan *password* digunakan untuk autentikasi ketika mengakses server *proxy*. *location* menunjukkan lokasi fisik atau geografis dari server. *admin_contact* menyimpan alamat email dari administrator yang bertanggung jawab atas pengelolaan server. *system_operation* menyatakan sistem operasi yang digunakan oleh server *proxy*, dengan nilai default "FreeBsd". *created_at* merekam waktu pertama kali data server dibuat, sementara *updated_at* diperbarui secara otomatis setiap kali ada perubahan data pada server.

Model *AccessLog* menyimpan catatan akses yang terjadi melalui *Squid Proxy* dengan berbagai informasi terkait permintaan yang dibuat oleh klien. timestamp menyimpan waktu ketika permintaan dilakukan dalam format *string*. *elapsed_time* mencatat durasi waktu yang dibutuhkan untuk memproses permintaan dalam satuan milidetik. *client_address* menyimpan alamat IP klien yang mengakses *server proxy*. *http_status* mencatat status HTTP yang dikembalikan oleh server sebagai respons terhadap permintaan. *bytes* menyimpan jumlah data dalam *byte* yang dikirimkan dari server ke klien. *request_method* mencatat metode HTTP yang digunakan dalam permintaan, seperti GET atau POST. *request_url* berisi URL tujuan yang diakses oleh klien melalui *proxy*. *host* menyimpan informasi tentang host yang diminta dalam permintaan, yang dapat bersifat opsional. server adalah relasi dengan model *ProxyServerInfo*, menunjukkan server mana yang menangani permintaan tersebut.

4.6 Pengkodean Otomatisasi SSH

Adapun hasil pengkodean Otomatisasi SSH, diperlihatkan pada gambar 4.3 dibawah ini.

```python

class UpdateDataLogProxy(AsyncWebsocketConsumer):
    async def connect(self):
        await self.accept()
        self.periodical = asyncio.create_task(self.send_periodic_updates())

    async def disconnect(self, code):
        self.periodical.cancel()

    async def send_periodic_updates(self):
        try:
            while True:
                # Buat satu koneksi SSH
                ssh_client = paramiko.SSHClient()
                ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
                server = await sync_to_async(ProxyServerInfo.objects.get)(id=1)

                ssh_client.connect(
                    hostname=server.ip_address,
                    port=22,
                    username=server.username,
                    password=server.password,
                )

                try:
                    # Jalankan semua update menggunakan koneksi yang sama
                    await asyncio.gather(
                        self.update_periodic_access(ssh_client, server),
                        self.update_periodic_store(ssh_client, server),
                        self.update_periodic_useragent(ssh_client, server),
                        self.update_periodic_cache(ssh_client, server),
                    )
                finally:
                    ssh_client.close()

                # Kirim pesan WebSocket (opsional)
                await self.send(
                    text_data=json.dumps({"message": "Periodic updates completed"})
                )

                # Tunggu sebelum mengulang
                await asyncio.sleep(600)  # 10 menit

        except asyncio.CancelledError:
            pass
        except Exception as e:
            await self.send(
                text_data=json.dumps({"message": f"Periodic update failed: {str(e)}"})
            )
```

Gambar 4. 3 Hasil Pengkodean Otomatisasi SSH

Kode *UpdateDataLogProxy* pada gambar diatas adalah implementasi *WebSocket* consumer berbasis asinkron yang digunakan untuk mengirimkan pembaruan data secara periodik melalui koneksi *WebSocket*. Saat koneksi *WebSocket* dibuka (metode *connect*), akan memulai task asinkron bernama

send_periodic_updates, yang bertugas melakukan koneksi SSH ke server menggunakan *library paramiko*. Informasi server diambil dari *database Django* menggunakan *ProxyServerInfo* secara asinkron dengan bantuan *sync_to_async*. Setelah koneksi SSH berhasil, task ini menjalankan beberapa fungsi pembaruan data secara paralel menggunakan *asyncio.gather*, yaitu *update_periodic_access*. Hasil pembaruan kemudian dikirimkan kembali ke klien *WebSocket*. Proses ini diulang setiap 10 menit menggunakan await *asyncio.sleep*(600). Jika koneksi *WebSocket* terputus, metode *disconnect* akan membatalkan *task* periodik tersebut untuk menghentikan pembaruan. Jika terjadi kesalahan selama proses, pesan error dikirimkan melalui *WebSocket* agar klien dapat mengetahui kegagalannya.

4.7 Hasil Black Box Testing

Pada pengujian ini terdapat terdapat 9 skenario pengujian. Hasil pengujian tedapat pada tabel 4.5 dibawah ini.

Tabel 4. 5 Hasil Black Box Testing

Test ID	Skenario	Hasil	Ket
TC01	Koneksi *WebSocket* berhasil	*WebSocket* terhubung dengan sukses.	Lulus
		Pesan sukses diterima di klien (jika ada).	
TC02	Koneksi *WebSocket* ditutup	Koneksi terputus tanpa *error*.	Lulus
		Tugas periodik dibatalkan tanpa masalah.	
TC03	Pengiriman pesan periodik berhasil	*WebSocket* mengirim pesan periodik berhasil.	Lulus
TC04	Kesalahan pada periodik	Pesan error dikirim ke klien melalui *WebSocket*.	Lulus
TC05	Data log diperbarui ke *database*	Data baru dari log *Squid* (*access.log*) disimpan ke *database*.	Lulus
		Pesan *WebSocket*: data terakhir yang valid.	
TC06	*Log file* tidak ditemukan	Tidak ada pesan *error* dikirim ke klien.	Tidak Lulus
		crash pada WebSocket atau server.	
TC07	*WebSockes* tetap aktif saat koneksi klien terputus	WebSocket server menangani dengan benar tanpa *error*.	Lulus
		Fungsi *disconnect*() dipanggil untuk membatalkan tugas periodik.	
TC08	Data yang sudah ada tidak disimpan ulang ke database	Data yang sudah ada di database tidak disimpan ulang.	Lulus

		Tidak ada data duplikat di *database*.	
TC09	Koneksi *WebSocket* dalam mode beban tinggi	WebSocket tetap stabil tanpa *error*.	Lulus
		Semua klien menerima pembaruan periodik sesuai interval.	

Pengujian otomatisasi SSH dilakukan dalam 9 skenario. Dari keseluruhan skenario tersebut, terdapat satu skenario yang tidak sesuai harapan, yaitu skenario berkode TC06. Kegagalan ini terjadi karena WebSocket mati secara langsung tanpa memberikan pesan kesalahan apabila terjadi *error* pada *library Paramiko*. *Error* tersebut disebabkan oleh kehilangan koneksi SSH pada *proxy server*. Untuk mengatasi masalah ini, solusi yang dapat diterapkan dalam penelitian ini adalah dengan memulai ulang aplikasi *Django*. Temuan ini menunjukkan bahwa dalam penggunaan *Paramiko* untuk komunikasi melalui proxy server, diperlukan mekanisme tambahan untuk menangani gangguan koneksi secara lebih efektif.

4.7 Analisis Efesiensi Otomatisasi SSH

Adapun hasil perbandingan pegambilan data dengan teknologi otomatisasi SSH dibandingkan dengan cara manual, diperlihatkan pada tabel 4.6.

Tabel 4. 6 Perbandingan Data Teknologi Otomatisasi SSH dibandingkan dengan cara manual

Waktu (menit)	Data *Log* Dari Otomatisasi SSH	Data *Log* Dari Metode Manual
1	600	6
5	3,000	30
10	6,000	60
20	12,000	120
30	18,000	180
40	24,000	240
50	30,000	300
60	36,000	300

Tingkat efisiensi *WebSocket* dalam mendapatkan data *log cache proxy* dapat dianalisis dengan membandingkan kecepatan dan jumlah data yang diperoleh melalui sistem otomatis berbasis *WebSocket* dengan metode manual. Berdasarkan simulasi data yang telah dijelaskan sebelumnya, sistem *monitoring cache proxy* berbasis *WebSocket* mampu mengumpulkan hingga 36.000 log dalam durasi satu jam, sementara metode manual hanya mampu mencatat sekitar 300 log dalam waktu yang sama, dengan asumsi operator bekerja secara efektif selama 50 menit dan mengambil istirahat selama 10 menit. Perbedaan yang signifikan ini menunjukkan bahwa sistem berbasis *WebSocket* memiliki efisiensi yang jauh lebih tinggi, yakni sekitar 120 kali lebih cepat dibandingkan metode manual.

Keunggulan utama *WebSocket* terletak pada kemampuannya untuk menyediakan komunikasi dua arah secara *real-time* antara klien dan server. Dalam konteks *monitoring cache proxy*, *WebSocket* memungkinkan data log, seperti access log, dikirimkan secara kontinu tanpa perlu melakukan *polling* berkala, yang biasanya memakan waktu lebih lama dan membebani jaringan. Hal ini berbeda dengan metode manual, di mana operator harus mengakses server *proxy* melalui terminal, mengambil data log secara bertahap, dan mencatatnya secara manual ke dalam database. Proses manual ini tidak hanya memakan waktu lebih lama tetapi juga rentan terhadap kesalahan manusia, seperti pencatatan data yang tidak lengkap atau tidak akurat.

Selain itu, penggunaan *WebSocket* yang terintegrasi dengan *Django* dan *Paramiko* memberikan fleksibilitas untuk memanfaatkan SSH sebagai jalur komunikasi aman dengan *server proxy*. Sistem ini memungkinkan pengambilan data log secara otomatis dari berbagai *server proxy* yang terhubung tanpa memerlukan intervensi manual. Hasilnya, data log yang diterima selalu aktual dan dapat

diakses secara *real-time*, sehingga meningkatkan kecepatan respons terhadap permasalahan yang terjadi pada *server proxy*.

Analisis ini menunjukkan bahwa tingkat efisiensi *WebSocket* tidak hanya terletak pada kecepatan pengambilan data, tetapi juga pada kemampuannya untuk memastikan data log tetap konsisten, real-time, dan terintegrasi langsung ke dalam sistem *monitoring*. Dalam skenario di mana data *log cache proxy* harus dipantau secara intensif untuk mendukung pengambilan keputusan, penggunaan *WebSocket* memberikan keunggulan yang tidak dapat dicapai oleh metode manual, sehingga menjadikannya solusi yang sangat efektif dan efisien.

5. KESIMPULAN

Berdasarkan hasil dari penelitian pada bab sebelumnya, didapatkan kesimpulan sebagai berikut:

1. Pengembangan teknologi otomatisasi SSH pada server *proxy* dengan *Django* berhasil diselesaikan menggunakan metode RAD, dengan tingkat keberhasilan pengujian sebesar 88%. Kegagalan pada iterasi pertama berhasil diperbaiki pada iterasi kedua, sehingga seluruh kebutuhan fungsional dan non-fungsional terpenuhi.
2. *WebSocket* mampu mengumpulkan hingga 36.000 log per jam, dibandingkan dengan metode manual yang hanya mencatat sekitar 300 log per jam, menjadikannya 120 kali lebih cepat.
3. Berdasarkan hasil implementasi dan pengujian, ditemukan bahwa library Paramiko memiliki keterbatasan dalam menangani kesalahan ketika sambungan proxy server terputus, di mana library ini tidak dapat memberikan pesan kesalahan dan langsung berhenti. Oleh karena itu, solusi yang dapat diterapkan dalam penelitian ini adalah dengan memulai ulang aplikasi Django. Temuan ini menunjukkan bahwa dalam spenggunaan Paramiko untuk komunikasi melalui proxy server, diperlukan mekanisme tambahan untuk menangani gangguan koneksi secara lebih efektif.

UCAPAN TERIMA KASIH

Penulis ingin menyampaikan rasa terima kasih yang mendalam kepada semua pihak yang telah memberikan dukungan dan bantuan selama penelitian ini. Dukungan dari berbagai pihak sangat berarti dan berperan penting dalam kelancaran serta keberhasilan penelitian ini. Terima kasih atas kontribusi dan partisipasi yang berharga dalam proses penelitian ini.

DAFTAR PUSTAKA

[1] Brian D. Davison, “A Web Caching Primer,” *IEEE Internet Computing,* vol. 5, no. 4, pp. 38-45, Agustus 2001.

[2] Kukuh Nugroho, Anggi Dzikri Abrariansyah, Syariful Ikhwan, “Perbandingan Kinersja Library Paramiko dan Netmiko Dalam Proses Otomasi Jaringan,” *Jurnal Nasional Informatika dan Teknologi Jaringan,* vol. 5, no. 1, pp. 1-8, 2020.

[3] Jake Kronika, Aidas Bendoraitis, Django 2 Web Development Cookbook ThirdEdition, Birmingham: Packt Publishing, Oktober 2018.

[4] Saini, Qulbir, Squid Proxy Server, Birmingham: Packt Publishing Ltd., 2011.

[5] Anindya Datta, Kaushik Dutta, Helen Thomas, Debra VanderMee, “World Wide Wait: A Study of InternetScalability and Cache-Based Approaches to Alleviate It,” *Management Science,* vol. 49, no. 10, pp. 1426-1444, Oktober 2003.

[6] Daniel J. Barrett, Richard Silverman, SSH, The Secure Shell: The Definitive Guide, Sebastopol, CA, USA: O’Reilly Media, Inc., Januari 2001.

[7] K. A. Nugraha, “Efisiensi Pertukaran Data Client-Server Menggunakan Web Socket pada Perangkat Berbasis Internet of Things,” *Jurnal Edukasi Dan Penelitian Informatika,* vol. 18, no. 4, pp. 33-39, 2024.

[8] Qigang Liu, Xiangyang Sun, “Research of Web Real-Time Communication Based on Web Socket,” *International Journal of Communications, Network and System Sciences,* vol. 5, no. 12, pp. 798-801, Oktober 2012.

[9] Jeff Forcier, Paul Bissex, Wesley Chun, Python Web Development With Django, London: Addison-Wesley, Oktober 2009.

[10] Drs. Afrizal zein M.Kom, Ir. Dahlan Susilo, M.Kom, Mustakim, M. kom, Ryan Effendi, Winny Purbaratri M.Kom, Achmad Ridwan, S.T., M.Si, Subhan Nooriansyah, S.Kom., M.Kom, Faridatun Nadziroh, S.ST., MT, Anyan, S.Kom., M.Kom., Dr. Ali Ibrahim. S.Kom, M.T, Konsep Dasar Rekayasa Perangkat Lunak, Kota Batam: Penerbit Yayasan Cendikia Mulia Mandiri, April 2023.

[11] M. E. Khan, "Different Approaches to Black Box Testing Technique for Finding Errors," *International Journal of Software Engineering and its Applications,* vol. 2, no. 4, pp. 31-40, 2011.

[12] Bayu Aji Priyaungga, Dwi Bayu Aji, Mukron Syahroni, Nurul Tri Sukma Aji, Aries Saifudin, "Pengujian Black Box pada Aplikasi Perpustakaan Menggunakan Teknik Equivalence Partitions," *Jurnal Teknologi Sistem Informasi dan Aplikasi,* vol. 3, no. 3, pp. 148-157, Juli 2020.