RANCANG BANGUN REST API DAN OTOMATISASI SSH PADA *PROXY SERVER* MENGGUNAKAN *FRAMEWORK DJANGO* UNTUK MENDUKUNG SISTEM *MONITORING CACHE PROXY*

(STUDI KASUS: PT. QUEEN NETWORK NUSANTARA)

(Skripsi)

Oleh

M. NAWWIR ALBI

PROGRAM STUDI TEKNIK INFORMATIKA
JURUSAN TEKNIK ELEKTRO
FAKULTAS TEKNIK
UNIVERSITAS LAMPUNG
2025

RANCANG BANGUN REST API DAN OTOMATISASI SSH PADA *PROXY SERVER* MENGGUNAKAN *FRAMEWORK DJANGO* UNTUK MENDUKUNG SISTEM *MONITORING CACHE PROXY*

(STUDI KASUS: PT. QUEEN NETWORK NUSANTARA)

Oleh

M. NAWWIR ALBI

Skripsi

Sebagai Salah Satu Syarat untuk Mencapai Gelar
SARJANA TEKNIK

Pada

Program Studi Teknik Informatika
Fakultas Teknik

PROGRAM STUDI TEKNIK INFORMATIKA
JURUSAN TEKNIK ELEKTRO
FAKULTAS TEKNIK
UNIVERSITAS LAMPUNG
2025

ABSTRAK

RANCANG BANGUN REST API DAN OTOMATISASI SSH PADA *PROXY SERVER* MENGGUNAKAN *FRAMEWORK DJANGO* UNTUK MENDUKUNG SISTEM *MONITORING CACHE PROXY*

(STUDI KASUS: PT. QUEEN NETWORK NUSANTARA)

Oleh

M. NAWWIR ALBI

Saat ini perusahaan PT. Queen Network Nusantara, proses *monitoring* dilakukan secara manual dengan memeriksa server *proxy* satu per satu melalui terminal atau SSH (*Secure Shell*) untuk memperoleh data-data tersebut. Metode ini tidak hanya kompleks tetapi juga memakan waktu sekitar 5 menit untuk mendapatkan dan mencatat semua data log *cache proxy* dari setiap server. Untuk mengatasi kendala ini, PT. Queen Network Nusantara membutuhkan sistem monitoring cache proxy berbasis web. Sistem ini diharapkan dapat mempermudah staf dalam melakukan monitoring dengan menampilkan data log cache proxy secara aktual, tanpa perlu mengakses terminal proxy server. Tujuan dari penelitian ini adalah untuk merancang dan membangun *REST API dan* 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) untuk pengembangan sistem yang cepat dan fleksibel. Tahap *requirement planning* dilakukan melalui studi literatur dan wawancara, *user design* menggunakan ERD dan desain *endpoint* API, *construction* dengan *Framework Django*, dan pengujian fungsionalitas dan non fungsionalitas menggunakan *Blackbox*. Hasil pengujian menunjukkan delapan fitur utama berhasil dikembangkan dengan 29 skenario dengan tingkat keberhasilan pengujian sebesar 92%, 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.

Kata kunci: *Cache*, *Django*, *Website*, *Proxy Server*

ABSTRACT

REST API DESIGN AND SSH AUTOMATION ON PROXY SERVER USING DJANGO FRAMEWORK TO SUPPORT PROXY CACHE MONITORING SYSTEM

(CASE STUDY: PT. QUEEN NETWORK NUSANTARA)

By

M. NAWWIR ALBI

Currently, at PT. Queen Network Nusantara, the monitoring process is conducted manually by checking proxy servers one by one through the terminal or SSH (Secure Shell) to retrieve cache proxy log data. This method is not only complex but also time-consuming, taking approximately 5 minutes to collect and record all cache proxy log data from each server. To address this issue, PT. Queen Network Nusantara requires a web-based cache proxy monitoring system. This system is expected to simplify the monitoring process by displaying real-time cache proxy log data without needing to access the proxy server terminal. The objective of this research is to design and develop a REST API and automate SSH for proxy servers to support the cache proxy monitoring system at PT. Queen Network using the Django framework. The research adopts the Rapid Application Development (RAD) method to ensure a fast and flexible system development process. The requirement planning phase was carried out through literature studies and interviews, user design was created using ERD dan desain *endpoint* API*, construction was implemented with the Django framework, and functionality and non-functionality testing were performed using the Blackbox method. The testing results showed that eight main features were successfully developed, covering 29 scenarios with a success rate of 92%. Additionally, the WebSocket testing demonstrated the ability to collect up to 36,000 logs per hour, compared to the manual method, which recorded only around 300 logs per hour, making it 120 times faster.*

keywords: *Cache*, *Django*, *Website*, *Proxy Server*

Judul Skripsi	:	**RANCANG BANGUN REST API DAN OTOMATISASI SSH PADA *PROXY SERVER* MENGGUNAKAN *FRAMEWORK DJANGO* UNTUK MENDUKUNG SISTEM *MONITORING CACHE PROXY***
Nama Mahasiswa	:	M. Nawwir Albi
Nomor Pokok Mahasiswa	:	205061081
Program Studi	:	S1 Teknik Informatika
Fakultas	:	Teknik

Menyetujui

1. Komisi Pembimbing

Pembimbing Utama	Pembimbing Pendamping
Yessi Mulyani, S.T., M.T.	**Puput Budi Wintoro, S. Kom, M.T.I**
NIP. 197312262000122001	**NIP. 198410312019031004**

2. Mengetahui

Ketua Jurusan Teknik Elektro	Ketua Program Studi Teknik Informatika
Herlinawati, S.T., M.T.	**Yessi Mulyani, S.T., M.T.**
NIP. 197103141999032001	**NIP. 197312262000122001**

MENGESAHKAN

1. Tim Penguji

Ketua : **Yessi Mulyani, S.T., M.T.** .....

Sekretaris : **Puput Budi Wintoro, S. Kom, M.T.I** .....

Penguji : **Wahyu Eko Sulistiono, S.T., M.Sc.** .....

2. Dekan Fakultas Teknik

Dr. Eng. Ir. Helmy Fitriawan, S.T., M.Sc.
NIP. 19750928 200112 1 002

Tanggal Lulus Ujian Skripsi : **28 Februari 2025**

SURAT PERNYATAAN

Saya yang bertandatangan di bawah ini, menyatakan bahwa skripsi saya dengan judul "Rancang Bangun REST API Dan Otomatisasi SSH Pada *Proxy Server* Menggunakan *Framework Django* Untuk Mendukung Sistem *Monitoring Cache Proxy* (Studi Kasus: PT. Queen Network Nusantara)" dibuat oleh saya sendiri. Semua hasil yang tertuang dalam skripsi ini telah mengikuti kaidah penulisan karya ilmiah Universitas Lampung. Apabila di kemudian waktu skripsi ini terbukti merupakan salinan atau dibuat oleh orang lain, maka saya bersedia menerima sanksi sesuai dengan ketentuan hukum yang berlaku.

Bandar Lampung, 28 Februari 2025
Pembuat pernyataan,

M. Nawwir Albi
NPM. 2015061081

RIWAYAT HIDUP

Penulis lahir di Bandar Lampung, pada tanggal 13 Januari 2002. Penulis merupakan

anak pertama dari pasangan suami istri Bapak Nabhan dan Ibu Sahelna. Penulis menyelesaikan pendidikannya di SDN 1 Pesawahan pada tahun 2014, MTSN 1 Bandar Lampung pada tahun 2017, dan MAN Insan Cendekia OKI pada tahun 2020. Bertepatan pada tahun itu juga, penulis terdaftar sebagai mahasiswa Program Studi Teknik Informatika, Jurusan Teknik Elektro, Fakultas Teknik Universitas Lampung melalui jalur SBMPTN. Selama menjadi mahasiswa, penulis melakukan beberapa kegiatan, antara lain:

1. Menjadi anggota organisasi Himpunan Mahasiswa Teknik Elektro Universitas Lampung, Departemen Wira Usaha pada tahun 2020.
2. Menjadi anggota organisasi Himpunan Mahasiswa Teknik Elektro Universitas Lampung, Departemen Wira Usaha 2021.
3. Melaksanakan Porgram Kuliah Kerja Nyata di Desa Sukapura, Kecamatan Sukau, Kabupaten Lampung Barat, Provinsi Lampung pada bulan Januari 2023.
4. Melakukan program kerja praktik di PT. Karakatau Steel pada bulan Oktober sampai November tahun 2023 dengan tugas akhir kerja praktik membuat *Website Management System* (WMS).

MOTTO

"Janganlah engkau bersedih, sesungguhnya Allah bersama kita"

(Q.S. Al-Baqarah : 286)

"Manusia pesimis cendrung memilih menyerah dari pada berusaha sampai akhir dalam meraih kesuksesan, akan tetapi orang yang optimis tidak akan menyerah walaupun sedang di hujani beribu kesulitan demi meraih kesuksesan"

(Ali Bin Abi Thalib)

"Jangan pernah takut gagal, karena dari kegagalan kita akan menemui jawaban untuk menjadi lebih baik"

(BJ Habibie)

"Selama masih ada pemenang, maka akan selalu ada pecundang untuk melengkapinya"

(Madara Uciha)

PERSEMBAHAN

Bismillaahirrohmaanirrahiim,
Dengan mengharapkan ridho dari Allah SWT,
Dengan penuh rasa syukur dan ketulusan, karya tulis ilmiah ini kupersembahkan kepada dengan setulus hati:
Kedua orang tuaku tercinta, yang dengan kasih sayang, doa, dan pengorbanan tiada henti selalu menjadi pilar kekuatanku. Terima kasih atas segala cinta dan dukungan yang tanpa batas,
Para guru dan dosen, yang telah membimbing dan mencerahkan jalan ilmu ini. Setiap nasihat dan ilmu yang diberikan adalah cahaya yang tak ternilai dalam perjalanan hidupku,
Sahabat-sahabat seperjuangan, yang selalu menemani di setiap langkah, berbagi semangat, tawa, dan dukungan dalam suka maupun duka.
Dan
Diriku sendiri, atas keteguhan hati, kerja keras, dan kesabaran dalam menyelesaikan karya ini. Semoga ini menjadi langkah awal menuju masa depan yang lebih baik.

Terimakasih untuk segalanya,
Kalian adalah hartaku yang paling berharga

KATA PENGANTAR

Puji dan syukur kehadirat Allah SWT yang telah melimpahkan rahmat, inayah, dan hidayat-Nya, sehingga penulis bisa menyelesaikan penelitian sekaligus tugas akhir ini yang berjudul "Rancang Bangun REST API Dan Otomatisasi SSH Pada *Proxy Server* Menggunakan *Framework Django* Untuk Mendukung Sistem *Monitoring Cache Proxy* (Studi Kasus: PT. Queen Network Nusantara)". Dalam upaya menyelesaikan penelitian dan penulisan tugas akhir ini penulis telah menerima dukungan baik secara moril maupun materil yang sangat berharg dari berbagai pihak. Oleh karena itu, penulis ingin mengucapkan rasa terima kasih sebesar-besarnya kepada semua pihak yang telah membantu, khususnya kepada:

1. Kedua orangtuaku tercinta dan seluruh sanak saudara yang selalu memberikan doa, motivasi bijak dan kasih sayang yang sangat berarti bagi penulis untuk menyelesaikan penelitian ini dengan lancar;
2. Bapak Dr. Eng. Helmy Fitriawan, S.T., M.Sc., selaku Dekan Fakultas Teknik Universitas Lampung;
3. Ibu Herlinawati, S.T., M.T., selaku Ketua Jurusan Teknik Elektro Universitas Lampung;
4. Ibu Yessi Mulyani, S.T., M.T., selaku Ketua Program Studi Teknik Informatika Universitas Lampung dan selaku Pembimbing Utama yang telah membantu proses pengerjaan penelitian dan memberikan banyak saran serta masukan terhadap penelitian ini;
5. Bapak Puput Budi Wintoro, S. Kom, M.T.I, selaku pembimbing pendamping yang telah membantu proses pengerjaan penelitian dan memberikan pemahaman, wawasan serta masukan terhadap penelitian ini;
6. Bapak Wahyu Eko Sulistiono, S.T., M.Sc., selaku Penguji yang telah memberikan banyak saran dan masukan terhadap penelitian ini, agar menjadi lebih baik lagi;

7. Bapak Puput Budi Wintoro, S. Kom, M.T.I, selaku Pembimbing Akademik yang memberikan bimbingan dan dukungan kepada penulis selama mengikuti jenjang perkuliahan;
8. Seluruh dosen dan staf Jurusan Teknik Informatika Unila yang memberi masukan dan mempermudah proses pembuatan skripsi/tugas akhir ini;
9. Teman-teman Mahasiswa Teknik Elektro Angkatan 2020 yang telah menjadi teman seperjuangan menempuh jenjang sarjana. Terimakasih telah menjadi teman penulis;

Laporan ini bisa jadi referensi penelitian maupun ide bisnis di bidang teknik informatika. Oleh karena itu, sebagai sesama peneliti diharapkan menggunakan berbagai refrensi sebagai mestinya.

Bandar Lampung, 2025
Penulis,

M. Nawwir Albi

DAFTAR ISI

Halaman

DAFTAR GAMBAR

Halaman

DAFTAR TABEL

Halaman

I. PENDAHULUAN

1.1 Latar Belakang

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 pemantauan lalu lintas jaringan pada *proxy* server untuk mencatat data *log cache proxy*. *Monitoring* ini

bertujuan untuk mendeteksi potensi masalah pada *cache proxy* dengan memanfaatkan data *log cache proxy*, seperti *access log*, *store log*, *cache log*, dan *user agent log*. Berdasarkan pengamatan penggunaan data-data *log cache proxy* di PT. *Queen Network* Nusantra, data-data tersebut dapat digunakan sebagai informasi penting untuk mendeteksi berbagai jenis masalah. *Access log*, misalnya, dapat mendeteksi akses yang mencurigakan atau tidak sah ke sumber daya jaringan. *User agent log* berguna untuk mengidentifikasi masalah spesifik yang terkait dengan aplikasi atau *browser* tertentu. *Cache log* membantu dalam mengidentifikasi kegagalan *cache proxy*. *Store log* memberikan informasi tentang konten yang sering di-*cache* atau dihapus, yang dapat membantu mengoptimalkan penggunaan penyimpanan.

Saat ini proses *monitoring* lalu lintas jaringan pada *proxy* server masih dilakukan secara manual dengan memantau data-data *log cache proxy* yang didapatkan 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 sistem monitoring *cache proxy* berbasis web. Sistem ini diharapkan dapat mempermudah staf dalam melakukan *monitoring* dengan menampilkan data *log cache proxy* secara aktual, tanpa perlu menulis perintah terminal secara langsung pada *proxy server*.

Sistem *monitoring cache proxy* dibagi menjadi dua bagian yang terpisah yakni *frontend* dan *back-end. Frontend* bertanggung jawab untuk memberikan antarmuka sistem *monitoring*. Sedangkan pada bagian *backend* bertanggung jawab untuk membuat layanan API (*Application Programming Interface*), otomatisasi SSH pada *proxy* server, dan pengelolaan *database*. Proses pengembangan sistem *monitoring* tersebut menggunakan metode RAD (*Rapid Application Development*) yang dilakukan selama 90 hari, karena perusahaan membutuhkan sistem informasi ini agar cepat diimplementasikan dilingkungan perusahaan.

Sistem *monitoring cache proxy* ini membutuhkan layanan API (*Application Programming Interface*) yang mengikuti arsitektur REST (*Representational State Transfer*) dan otomatisasi SSH pada proxy *server*. REST API ini berfungsi untuk menjembatani proses pertukaran informasi (*request* dan *response*) antara *back-end* dan *front-end*. Selain itu, otomatisasi SSH pada *proxy server* berfungsi untuk menulis perintah terminal pada *proxy server* dengan menggunakan *script* sehinga proses pencatatan data *log cache proxy* dapat di persingkat.

Untuk mewujudkan layanan pada paragraf sebelumnya, sistem *monitoring cache proxy* menggunakan *framework Django*, karena *framework Django* sendiri sudah memiliki paket yang mendukung pembuatan REST API yaitu dengan menggunakan paket *Django Rest Framework* dan mendukung otomatisasi SSH melalui *Python* dengan menggunakan paket *Paramiko. Paramiko* memiliki kemampuan untuk menjalankan *script* yang berisikan sebuah perintah konfigurasi pada *device*, kemudian perintah tersebut dikirimkan melalui protokol SSH (*Secure Shell*) 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 tanpa, sehingga proses pengambilan data *log cache proxy* bisa dilakukan secara *realtime* dan disimpan kedalam *database*. *Framework* tersebut menggunakan bahasa pemrograman *Python*, yang termasuk dalam kategori bahasa tingkat tinggi dan memiliki struktur yang mirip dengan bahasa manusia, sehingga mempermudah proses pengembangan. Maka dari itu, berdasarkan topik diatas yang berfokus pada pengembangan REST API dan Otomatisasi SSH pada *proxy server* dapat diangkat sebagai studi kasus penelitian yang berjudul "Rancang Bangun Rest API Dan Otomatisasi SSH Pada *Proxy Server* Menggunakan *Framework Django* Untuk Mendukung Sistem *Monitoring Cache Proxy*".

1.2 Tujuan Penelitian

Tujuan dari penelitian ini adalah untuk merancang dan membangun *REST API dan* otomatisasi SSH pada server *proxy* untuk mendukung sistem *monitoring cache proxy* di PT. *Queen Network* dengan menggunakan *framework Django.*

1.3 Rumusan Masalah

Rumusan pada penelitian ini antara lain :

1. Bagaimana tingkat keberhasilan pengujian fungsional dan non fungsional yang didapatkan dari implementasi REST API dan otomatisasi SSH pada server *proxy* untuk sistem *monitoring cache proxy* ?
2. Bagaimanakah tingkat keefektifan pengambilan data log secara *realtime* menggunakan *webscoket* dibandingkan dengan menggunakan metode manual ?

1.4 Batasan Masalah

Dalam penelitian ini, menetapkan batasan masalah ruang lingkup *deployment* hanya berada di jaringan lokal perusahaan.

1.5 Sistematika Penulisan

Sistematika penulisan skripsi (tugas akhir) ini terdiri dari 5 (lima) bab sebagai berikut:

BAB I : PENDAHULUAN

Bab berisikan latar belakang, tujuan penelitian, rumusan masalah, batasan masalah, dan sistematika penulisan.

BAB II : TINJAUAN PUSTAKA

Memaparkan teori-teori dasar yang digunakan sebagai referensi untuk memahami permasalahan terkait dalam melakukan penelitian meliputi *Proxy Server, Cache Proxy,* Otomatisasi SSH*,* REST API, *Websocket,* ERD (*Entity Relationship Diagram*)*, Rapid Aplication Development, dan Black Box Testing.*

BAB III : METODOLOGI PENELITIAN

Bab berisikan waktu dan tempat penelitian, alat dan bahan penelitian, dan tahapan penelitian.

BAB IV : HASIL DAN PEMBAHASAN

Dalam bab ini memuat hasil dan pembahasan yang didapat dalam penelitian ini.

BAB V : KESIMPULAN DAN SARAN

Dalam bab ini memuat kesimpulan dan saran dari hasil penelitian.

II. 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 [3].

Dalam bentuk yang lebih canggih, *proxy server* dapat memfilter permintaan berdasarkan berbagai aturan dan hanya mengizinkan komunikasi jika permintaan tersebut sesuai dengan aturan yang telah ditentukan. Aturan-aturan ini umumnya didasarkan pada alamat IP klien atau server target, protokol, jenis konten dokumen web, tipe konten web, dan sebagainya [3].

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* [3].

Caching adalah teknik yang populer digunakan oleh server proxy untuk menghemat bandwidth, meningkatkan kinerja server web, dan memberikan pengalaman penjelajahan yang lebih baik bagi pengguna akhir [3].

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 [4].

Untuk memahami hal ini, pertimbangkan struktur Internet yang terdiri dari sistem *end-device* yang terhubung ke penyedia layanan Internet (ISP) lokal. ISP lokal kemudian terhubung ke ISP regional, yang selanjutnya terhubung ke penyedia backbone nasional (NBP). NBP saling terhubung melalui titik akses jaringan (NAP) atau hubungan *peering* pribadi. *Caching* dapat terjadi di berbagai lokasi dalam struktur ini, seperti pada sistem pengguna akhir, ISP lokal atau regional, dan seterusnya [4].

Dengan menerapkan caching secara efektif, konten yang sering diakses dapat disajikan lebih cepat kepada *client*, mengurangi beban pada server asal, dan mengoptimalkan penggunaan bandwidth di seluruh jaringan. Hal ini meningkatkan kinerja jaringan secara keseluruhan dan memberikan pengalaman yang lebih baik bagi *client* [4].

2.3 Otomatisasi SSH

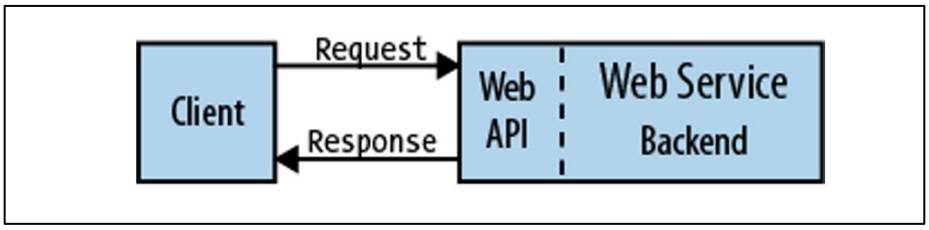
Otomatisasi SSH adalah proses menggunakan protokol *Secure Shell* (SSH) untuk menjalankan tugas secara otomatis pada sistem atau server jarak jauh tanpa interaksi manual yang berulang. Hal ini, ini melibatkan skrip atau alat yang

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, atau *Secure Shell*, adalah pendekatan berbasis perangkat lunak yang populer dan kuat untuk keamanan jaringan. Saat data dikirim oleh komputer ke jaringan, SSH secara otomatis mengenkripsi data tersebut. Ketika data tersebut mencapai penerima yang dituju, SSH secara otomatis mendekripsinya kembali. Proses ini disebut sebagai enkripsi transparan, di mana *client* dapat bekerja seperti biasa tanpa menyadari bahwa komunikasi telah dienkripsi dengan aman di jaringan. SSH menggunakan algoritma enkripsi *modern* yang aman dan cukup andal untuk digunakan dalam aplikasi penting di perusahaan besar [5].

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 [5].

2.4 *Rest API* (*Representational State Transfer Application Programming Interface*)

Server web yang dirancang khusus untuk mendukung kebutuhan sebuah situs atau aplikasi lainnya. Program klien menggunakan antarmuka pemrograman aplikasi (API) untuk berkomunikasi dengan layanan web. Secara umum, API menyediakan serangkaian data dan fungsi untuk memfasilitasi interaksi antara program komputer serta memungkinkan pertukaran informasi. Seperti yang digambarkan pada Gambar 2.1, API Web adalah antarmuka dari sebuah layanan web yang secara langsung mendengarkan dan merespons permintaan klien [6].



Gambar 2.1 *Web API*

Ada 4 prinsip utama REST yang ditetapkan oleh Roy Fielding dan rekan-rekannya pada tahun 2000. Mereka berupaya menciptakan standar yang memungkinkan server berkomunikasi dengan server lain dengan mudah. Inilah yang mereka hasilkan, yang mengubah lanskap API:

1. Klien-Server: Selalu ada klien dan server, dan kedua sistem ini memerlukan batasan untuk cara pengoperasiannya. Mana yang dipanggil (server) dan mana yang mengajukan permintaan (klien). Dengan adanya batasan ini, pengoperasian akan menjadi lebih lancar.
2. *Stateless*: Server harus dapat memproses pesan yang diterimanya. Untuk melakukan ini, setiap permintaan yang diterima server harus memiliki informasi yang diperlukan agar server dapat berfungsi.
3. Antarmuka Seragam: Penggunaan terminologi dan sumber daya yang serupa membantu menstandardisasi API. Berdasarkan prinsip ini, kata kerja HTTP berikut digunakan: GET, PUT, POST, dan DELETE. Sumber daya selalu merujuk ke URI (*uniform resource identifier*). Respons HTTP selalu disertai status.
4. Dapat disimpan dalam *cache*: Klien perlu dapat menyimpan representasi dalam *cache*. Karena tidak memiliki status (setiap representasi bersifat deskriptif sendiri), hal ini dimungkinkan dalam API RESTful.

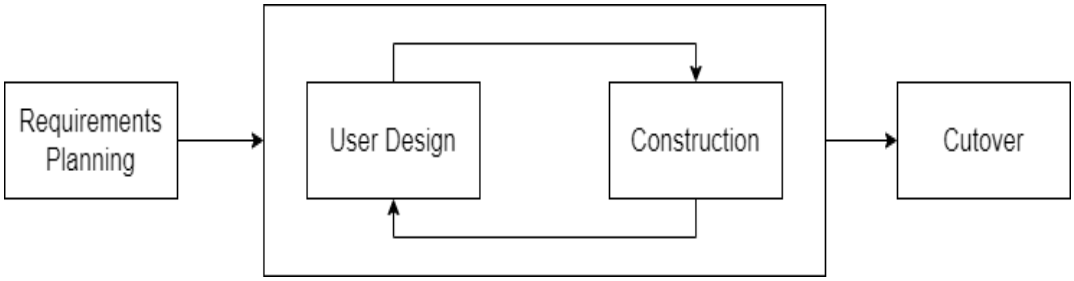
Gaya arsitektur REST sering digunakan dalam perancangan API untuk layanan web modern. API Web yang sesuai dengan gaya arsitektur REST disebut sebagai REST API [6].

2.5 *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].

2.6 Metode *RAD*

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 RAD (*Rapid Aplication Development*) 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 [9].



Gambar 2.2 Diagram RAD

Rapid Application Development (RAD) memiliki empat tahap yang terstruktur meliputi [9]:

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.

2.7 Metode *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. Pengujian 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. [10].

Black box testing juga dikenal dengan istilah lain seperti *opaque testing*, *functional testing*, *behavioral testing*, dan *closed-box testing*. Contoh sederhana dari sistem yang menggunakan metode ini adalah mesin pencari (*search engine*). Ketika kita memasukkan teks yang ingin dicari, kita mendapatkan hasilnya tanpa mengetahui atau melihat secara spesifik proses yang digunakan sistem untuk mendapatkan hasil tersebut [10].

Equivalence partitioning adalah metode *black box testing* yang membagi data masukan dari suatu unit perangkat lunak ke dalam beberapa partisi, dari mana kasus uji dapat diturunkan. Metode ini membantu mengurangi jumlah kasus uji yang diperlukan. Dalam *equivalence class partitioning*, kelas ekuivalen dibentuk berdasarkan masukan yang memiliki perilaku sistem yang serupa atau diharapkan serupa. Kelas ekuivalen ini mewakili sekumpulan keadaan valid atau tidak valid dari suatu kondisi masukan. Biasanya, kondisi masukan dapat berupa nilai numerik tertentu, kumpulan nilai, serangkaian nilai terkait, atau kondisi Boolean. Setelah kelas ekuivalen dipilih untuk setiap masukan, langkah berikutnya adalah menentukan kasus uji yang sesuai [10].

2.8 Penelitian Terkait

Adapun penelitian yang telah ditetapkan sebagai refrensi atau pedoman pada penelitian ini sebagai berikut.

2.8.1 PENGUJIAN *BLACK BOX* PADA APLIKASI PERPUSTAKAAN MENGGUNAKAN TEKNIK *EQUIVALENCE PARTITIONS*

Artikel jurnal penelitian berjudul “Pengujian Black Box pada Aplikasi Perpustakaan Menggunakan Teknik Equivalence Partitions ”. Diambil dari jurnal “Jurnal Teknologi Sistem Informasi dan Aplikasi”, diteliti oleh Bayu Aji

Priyaungga, Dwi Bayu Aji, Mukron Syahroni, Nurul Tri Sukma Aji, dan Aries Syafudin pada tahun 2023 membahas aplikasi perpustakaan yang berfungsi untuk mengelola dan mencatat berbagai transaksi, seperti peminjaman serta pengembalian buku. Agar aplikasi ini dapat berjalan tanpa kesalahan, artikel jurnal penelitian tersebut memerukan proses pengujian guna memastikan kualitasnya. Pengujian perangkat lunak bertujuan untuk mendeteksi serta mengidentifikasi bug atau kesalahan dalam sistem, sehingga dapat mengurangi potensi kerugian akibat kegagalan aplikasi [11].

Dalam penelitian ini, metode pengujian yang digunakan adalah *Black Box Testing*. Teknik ini berfokus pada pengujian tampilan antarmuka serta fungsi perangkat lunak tanpa mempertimbangkan kode sumbernya [11].

2.8.2 RANCANG BANGUN *BACK-END API* PADA *APLIKASI MOBILE AYAMHUB* MENGGUNAKAN FRAMEWORK NODE JS EXPRESS

Artikel jurnal penelitian berjudul " Rancang Bangun Back-end API pada Aplikasi Mobile AyamHub Menggunakan Framework Node JS Express ". Diambil dari jurnal "JUSTIN : Jurnal Sistem dan Teknologi Informasi ", diteliti oleh Eli Nurhayati dan Agussalim pada tahun 2023 membahas AyamHub adalah sebuah aplikasi yang dirancang untuk menghubungkan peternakan dengan pelaku UMKM atau penjual ayam broiler di Indonesia. Dalam pengembangan sistem ini, AyamHub memerlukan back-end yang akan menjadi dasar bagi seluruh operasional aplikasi. Untuk struktur back-end, aplikasi ini menggunakan arsitektur REST API. Proses pengembangan dilakukan dengan mengikuti metode *waterfall* [12].

2.8.3 ANALISIS PERBANDINGAN PERFORMA *RESTFUL API* ANTARA *EXPRESS.JS* DENGAN *LARAVEL FRAMEWORK* DENGAN *JMETER*

Artikel jurnal penelitian dengan judul "Analisis Perbandingan Performa *Restful API* Antara *Express* Dengan *Laravel Framework* Dengan *JMeter*". Diambil dari jurnal "Jurnal Informatika dan Teknik Elektro Terapan", diteliti oleh Wira Hadinata1, dan

Lilis Stianingsih pada tahun 2024 membahas perbandingan kinerja antara dua *framework backend*, Laravel dan Express.js, dilakukan untuk menilai waktu respon, penggunaan CPU, dan memori dalam pembuatan RESTful API [13].

2.8.4 PERANCANGAN DAN IMPLEMENTASI RESTFUL API PADA SISTEM INFORMASI MANAJEMEN DOSEN UNIVERSITAS UDAYANA

Artikel jurnal penelitian dengan judul "Perancangan Dan Implementasi RESTful API PADA Sistem Informasi Manajemen Dosen Universitas Udayana*"*. Diambil dari Jurnal SPEKTRUM, diteliti oleh Ida Ayu Kaniya Pradnya Paramitha, Dewa Made Wiharta, dan Made Arsa Suyadnya pada tahun 2022, membahas Sistem Informasi Manajemen (SIM) Dosen Universitas Udayana dirancang untuk membantu dosen meningkatkan kinerja dalam menyelenggarakan pendidikan tinggi yang berkualitas. Awalnya, sistem ini menggunakan arsitektur monolitik, namun kemudian beralih ke arsitektur microservice untuk memberikan fleksibilitas lebih dengan memisahkan beberapa aplikasi menjadi layanan terpisah. Komunikasi antar layanan dilakukan menggunakan protokol HTTP dengan RESTful API. Penelitian ini berfokus pada pengembangan RESTful API untuk modul data dosen, yang dibangun menggunakan bahasa pemrograman Java dan *database* MySQL. Pengujian fungsionalitas RESTful API dilakukan dengan pendekatan *black box* dan menggunakan aplikasi Postman pada perangkat lokal. Hasil pengujian menunjukkan bahwa metode GET, PUT, dan DELETE memberikan kode respons 200 dengan status OK, sementara metode POST menghasilkan kode respons 201 dengan status *Created*, yang menandakan bahwa semua fungsi metode HTTP pada modul data dosen telah berjalan dengan baik. [14].

2.8.5 IMPLEMENTASI *REST API WEB SERVICE* DALAM MEMBANGUN APLIKASI *MULTIPLATFORM* UNTUK USAHA JASA

Artikel jurnal penelitian yang berjudul "Implementasi Rest Api Web Service Dalam Membangun Aplikasi Multiplatform Untuk Usaha Jasa" oleh Romi Choirudin1, dan Ahmat Adil pada tahun 2020 membahas bagaimana perkembangan sektor

pariwisata di Pulau Lombok telah menciptakan beragam profesi baru, terutama di sektor jasa. Kemajuan teknologi telah mengubah cara masyarakat menjalankan usaha dan aktivitas mereka, dengan kebutuhan baru yang muncul, seperti kemudahan dalam mencari jasa tukang, mengetahui harga jasa, harga material, dan sebagainya. Aplikasi *multiplatform* dapat menawarkan fleksibilitas kepada pengguna untuk memilih platform yang paling sesuai bagi mereka [15].

Arsitektur web service pada aplikasi ini merujuk pada entitas komputasi yang dapat diakses melalui jaringan internet atau intranet dengan menggunakan protokol standar dan antarmuka bahasa pemrograman yang independen. Pengembangan sistem ini menggunakan metode waterfall yang bersifat sistematis dan berurutan, dimulai dari analisis, desain, implementasi (pengkodean), pengujian, hingga pemeliharaan sistem[15].

2.8.6 STUDI ANALISIS *RAPID APPLICATION DEVELOPMENT* SEBAGAI SALAH SATU ALTERNATIF METODE PENGEMBANGAN PERANGKAT LUNAK

Artikel jurnal penelitian dengan judul "Studi Analisis *Rapid Aplication Development* Sebagai Salah Satu Alternatif Metode Pengembangan Perangkat Lunak". Diambil dari jurnal " Jurnal Informatika ", diteliti oleh Agustinus Noertjahyana pada tahun 2002 membahas Rapid Application Development (RAD) merupakan salah satu metode alternatif dari System Development Life Cycle yang sering digunakan untuk mengatasi keterlambatan yang biasanya terjadi pada metode pengembangan tradisional. Keuntungan utama dari penggunaan metode ini adalah kecepatan, akurasi, serta biaya yang lebih rendah dibandingkan dengan metode konvensional. Selain itu, dengan melibatkan pengguna dalam proses perancangan, kebutuhan mereka dapat lebih mudah dipenuhi, yang pada gilirannya meningkatkan kepuasan pengguna terhadap sistem. Namun, dalam menerapkan metode RAD, perlu diperhatikan beberapa faktor penting, seperti kesiapan tim, ruang lingkup sistem, kebutuhan pengguna, dan kinerja sistem. Dengan demikian, RAD bisa menjadi pilihan yang efektif untuk mengembangkan sistem informasi yang dapat memenuhi kebutuhan pengguna. [16].

2.8.7 *EVALUATION UUSING BLACK BOX TESTING AND SYSTEM USABILITY SCALE IN THE KIDUNG SEKAR MADYA APPLICATION*

Artikel jurnal penelitian dengan judul "*Evaluation Using Black Box Testing and System Usability Scale in the Kidung Sekar Madya Application*". Diambil dari jurnal "Jurnal SINKRON", diteliti oleh Gede Surya Mahendra, I Kadek Andy Asmarajaya pada tahun 2022 mebahas tujuan untuk melestarikan budaya Kidung Dharma Gita dengan melakukan digitalisasi budaya Bali, khususnya lagu Dharma Gita Sekar Madya. Salah satu solusi yang ditawarkan adalah pengembangan aplikasi berbasis Android untuk Kidung Dharma Gita Sekar Madya. Evaluasi dilakukan dengan metode *Black Box Testing* dan *System Usability Scale* (SUS). [17].

2.8.8 *WEBSOCKET* UNTUK OPTIMASI KECEPATAN DATA TRANSFER PADA *REAL TIME CHATTING*

Artikel jurnal penelitian dengan judul "*Websocket* untuk Optimasi Kecepatan Data Transfer pada *Real Time Chatting*" termasuk dalam jurnal "*Innovation in Research of Informatics*" yang diteliti oleh Asep Rizki Maulana, Alam Rahmatulloh, pada tahun Maret 2019 merupakan penelitian yang membahas komunikasi *real-time* layaknya *chatting* dimana baru-baru ini menjadi populer dengan memfokuskan terhadap teknologi berbasis teks yang ditransmisikan melalui internet. Metode tradisional seperti AJAX, terutama long polling, memiliki kelemahan karena menyebabkan beban server meningkat saat banyak pengguna melakukan request. Artikel jurnal penelitian ini mengusulkan aplikasi *chatting* berbasis teknologi *WebSocket* yang mendukung komunikasi *full-duplex* untuk mengatasi masalah tersebut.[18].

2.8.9 PENGEMBANGAN APLIKASI OTOMATISASI ADMINISTRASI JARINGAN BERBASSIS *WEBSITE* MENGGUNAKAN BAHASA PEMROGRAMAN *PYTHON*

Artikel jurnal penelitian dengan judul "Pengembangan Aplikasi Otomatisasi Administrasi Jaringan Berbasis *Website* Menggunakan Bahasa Pemrograman *Python*" termasuk dalam jurnal "Jurnal SIMETRIS" yang diteliti oleh Rheza

Adhyatmaka Wiryawan, pada tahun Movember 2019 merupakan penelitian yang membahas perusahaan yang sedang menghadapi tantangan dalam mengelola banyak perangkat jaringan yang memerlukan pemeliharaan dan konfigurasi rutin agar jaringan tetap berjalan optimal. Metode tradisional yang mengharuskan administrator mengakses perangkat satu per satu dinilai kurang efisien. Oleh karena itu, otomatisasi jaringan pada artikel jurnal penelitian menjadi solusi untuk menangani tugas-tugas repetitif seperti *backup* dan *restore* konfigurasi secara lebih efektif [19].

2.8.10 PENGEMBANGAN APLIKASI AKREDITASI PROGRAM STUDI BERBASIS FRAMEWORK DJANGO

Penelitian dengan judul "Pengembangan Aplikasi Akreditas Program Studi Berbasis Framework Django ". Diambil dari jurnal "Jurnal Informatika", diteliti oleh Hary Sabita1 ,Riko Herwanto ,Yuli Syafitri, Bagus Dwi Prasetyo pada tahun 2022 membahas akreditasi program studi di perguruan tinggi, yang bertujuan untuk menilai kelayakan program studi dengan menggunakan instrumen IPEPA dari BAN PT yang dirilis pada tahun 2020. Kendala muncul terutama bagi perguruan tinggi yang tidak memiliki basis data sistematis, sehingga kurang memiliki rekam jejak data untuk referensi [20].

2.9 *State of The Art*

State of The Art merupakan poin-poin hasil analisa yang diperoleh dari penelitian terdahulu yang dijadikan bahan refrensi pada penelitian saat ini. Hasil analisis ini berguna untuk memberikan referensi, wawasan dan batasan pada penelitian yang sedang di lakukan. Berikut *state of the art* dari penelitian sebelumnya.

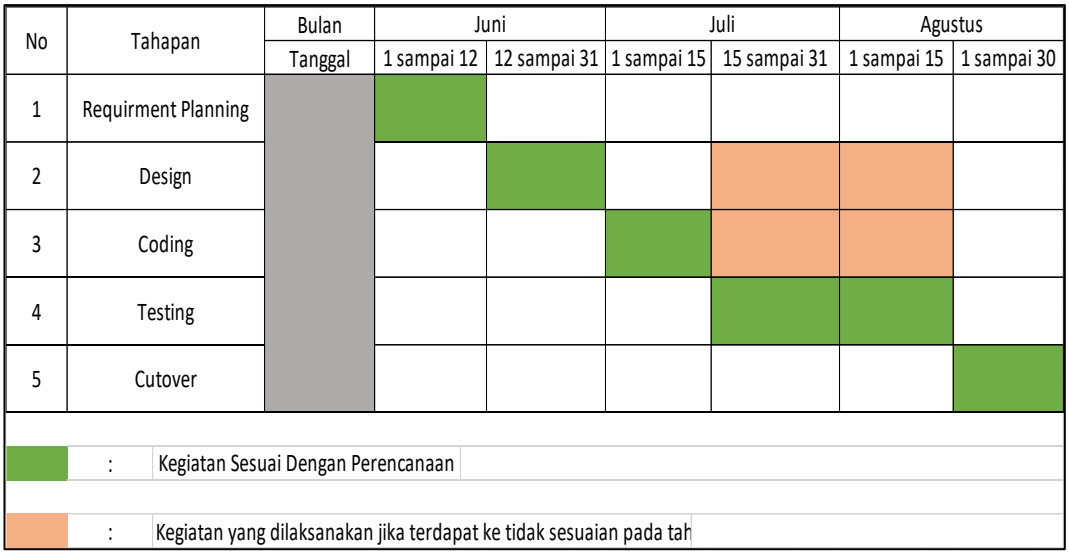
1. Berdasarkan penelitian [11], memberikan informasi sistematis bagaimana membangun aplikasi dengan *framework django* agar sesuai dengan kebutuhan sistem yang valid.
2. Berdasarkan penelitian [12], memberikan informasi sistematis bagaimana menguji dan menganalisis aplikasi dengan menggunakan *black box testing*.

3. Berdasarkan penelitian [13], memberikan informasi sistematis bagaimana perbandingan merancang dan membangun REST API selain menggunakan *framework django*.
4. Berdasarkan penelitian [14], memberikan informasi sistematis bagaimana menguji dan menganalisis performa dari layanan Restful API.
5. Berdasarkan penelitian [15], memberikan informasi sistematis bagaimana cara untuk merancang dan membangun layanan Restful API.
6. Berdasarkan penelitian [16], memberikan informasi bagaimana sekanario yang dibutuhkan untuk menguji *endpoint* Restful API dengan menggunakan Postman.
7. Berdasarkan penelitian [17], memberikan informasi desain UML apa sajakah yang dibututhkan untuk membangun *endpoint* Restful API dengan menggunakan Postman.
8. Berdasarkan penelitian [18], memberikan informasi lengkap tentang mentode pengembangan RAD (Rapid Aplication Development).
9. Berdasarkan penelitian [19], memberikan informasi dasar terkait proxy server.
10. Berdasarkan penelitian [20], memberikan informasi tentang bagaimana pengembangan aplikasi web *django dan paramiko* untuk otomatisasi ssh melalui perangkat jaringan seperti *switch*.

III. METODOLOGI PENELITIAN

3.1 Waktu dan Tempat

Penelitian dilaksanakan mulai dari bulan Juni 2024 sampai dengan bulan Agustus 2024 yang bertempat di kantor PT. Queen Network Nusantara. Dibawah ini Gambar 3.1 berisikan tahapan waktu penelitian.



Gambar 3.1 Tahapan Waktu Penelitian

3.2 Alat dan Bahan Penelitian

3.2.1 Alat Penelitian

Adapun alat yang digunakan pada penelitian ini dapat dilihat pada **Error! R eference source not found.**3.2 di bawah ini, sebagai berikut:

Tabel 3.1 Alat Penelitian

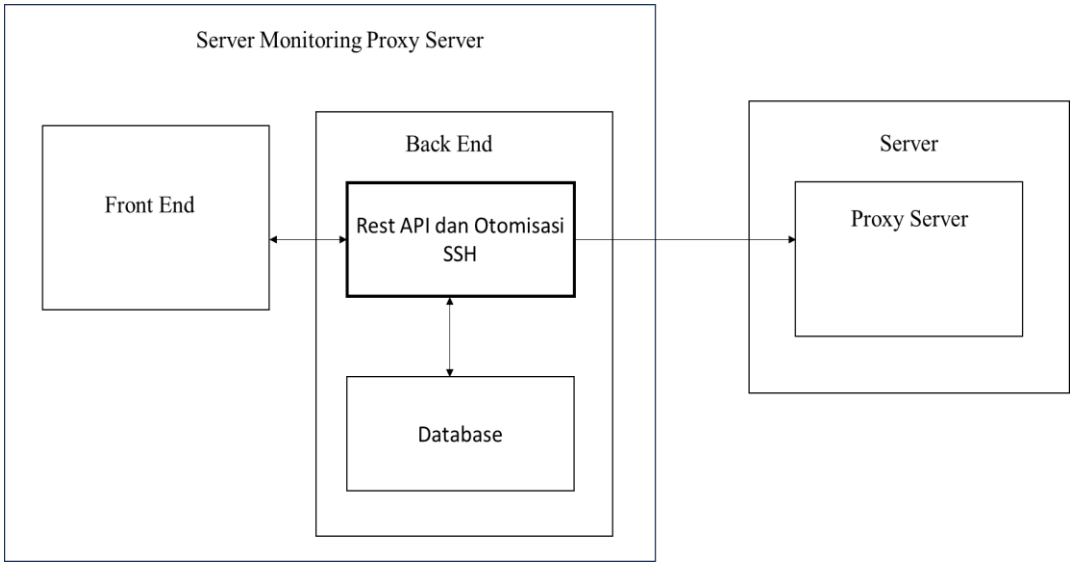
No.	Nama Alat	Spesifikasi	Deskripsi

1.	Laptop	Prosesor Intel Core i5-8250U, *Memory* 12GB, VGA *Nvidia GeForce* MX 130, *Windows* OS	Perangkat keras untuk membangun aplikasi.
2.	*Visual Studio Code*	Aplikasi *Desktop*, Versi 1.78.2	Perangkat lunak untuk menuliskan baris kode aplikasi
3.	*Django*	*Framework,* Versi 5.0.3	*Framework* untuk membangun *REST API dan Otomisasi SSH*
4	*Postman*	Aplikasi *Desktop*, Versi 2.3.16	*Tools* yang dapat digunakan untuk pengujian *REST API*
5	*Firefox*	Aplikasi *Desktop*, Versi 132.0.1	*Tools* yang dapat digunakan untuk menguji otomatisasi SSH melalui *websocket*
6	*Websocket Tester*	*Website*, Versi 3	*Tools* yang dapat digunakan untuk menguji otomatisasi SSH melalui *websocket*

3.2.2 Bahan Penelitian

Bahan yang digunakan dalam penelitian adalah *proxy server* PT. Queen Network Nusantara.

3.3 *Capstone Project*



Gambar 3.2 *Capstone Project*

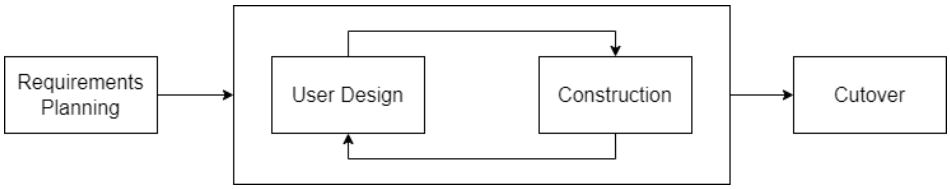
Gambar 3.1 merupakan gambar *Capstone Project* sistem *monitoring cache* pada *Proxy Server*. Pengembangan sistem *monitoring cache* pada *Proxy Server* dimulai dengan implementasi dan konfigurasi pada *proxy server*, berlanjut ke pengembangan *backend*, dan diakhiri dengan integrasi *frontend*.

- Pertama, tim *Proxy Server* bertanggung jawab untuk menginstal dan mengonfigurasi *Squid Web Cache* di sistem operasi *FreeBSD*. *Squid* digunakan untuk menangani *cache hit* dan *cache miss* dari objek yang sering diakses oleh pengguna. *Squid* dikonfigurasi untuk menghasilkan berbagai log, seperti *access log, store log, user agent log, dan cache log* yang berfungsi untuk mencatat aktivitas *proxy*. Setelah konfigurasi selesai, dilakukan pengujian untuk memastikan bahwa sistem mampu menangani beban lalu lintas jaringan dengan optimal dan menyimpan log dengan benar.
- Selanjutnya, tim *Backend* pada bagian *Rest API* dan Otomisasi *SSH* bertanggung jawab untuk mengembangkan Rest API untuk proses pengambilan, pemrosesan, dan penyajian data *cache* secara terstruktur, sehingga memudahkan tim *front-end* berkomunikasi dan mendapatkan informasi *cache* secara langsung. Tim ini juga, bertanggung jawab untuk mengengembangkan otomatisasi SSH yang digunakan untuk menangkap

serangkaian data log yang ada di *proxy server.* Perlu diingat bahwa bagian ini merupakan penelitian yang sedang di lakukan.

- Selanjutnya tim *backend* pada bagian *database* bertanggung jawab untuk mengutilisasi *database* agar data yang disimpan terorganisir dengan baik , mengoptimalkan *database*, dan menyesuaikan keamanan *database.* Sistem database menggunakan *PostgreSQL.*
- Setelah API siap, tim *Frontend* mengembangkan antarmuka, dengan tujuan menampilkan data *real-time* dari API dalam bentuk grafik dan tabel. *Frontend* mengintegrasikan API untuk menampilkan data log secara dinamis, dengan *WebSockets* untuk memperbarui data secara *real-time*. Antarmuka dirancang agar intuitif dan mudah digunakan, menyediakan fitur filter bagi pengguna untuk melihat data log berdasarkan parameter spesifik. *Dashboard* juga menyajikan bagan status *cache* dan jumlah log secara keseluruhan.

3.4 Tahapan Penelitian



Gambar 3.2 Tahapan Penelitian

Penelitian berjudul “Rancang Bangun Rest Api Dan Otomatisasi SSH Pada *Proxy Server* Menggunakan *Framework Django* Untuk Mendukung Sistem *Monitoring Cache Proxy*” dilaksanakan selama tiga bulan menggunakan model pengembangan perangkat lunak yang dikenal sebagai *Rapid Application Development* (RA*D*). Model RAD dipilih karena kemampuannya untuk mengakomodasi perubahan dan penyesuaian dengan cepat selama proses pengembangan.

3.4.1 *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 REST API dan 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*.

Pada tahapan wawancara ini ada beberapa pertanyaan yang diajukan kepada narasumber, yang dapat dijadikan sumber referensi dalam penelitian ini, pertanyaan tersebut antara lain. Di bawah ini Tabel 3.2 berisikan daftar pertanyaan dalam proses wawancara.

Tabel 3.2 Daftar Pertanyaan Wawancara

No.	Kode	Pertanyaan
1	P1	Seberapa penting menurut Anda *monitoring cache server* untuk kinerja keseluruhan sistem?
2	P2	Bagaimana selama ini perusahaan memantau performa *cache server*?
3	P3	Pernahkah ada mengalami masalah dengan performa *cache server* ? Jika ya, apa saja penyebabnya?
4	P4	Fitur apa yang saja yang diperlukan untuk sistem *monitoring cache server*?
5	P5	Bagaimanakah proses sistem ini , dalam transfer data ke *UI* sistem
6	P6	Apakah sistem ini perlu autentikasi pengguna?

7	P7	*Middleware* apakah yang digunakan aplikasi untuk mengamankan data pengguna ?
8	P8	*Framework* apakah yang digunakan untuk membangun keseluruhan sistem informasi ? dan mengapa ?
9	P9	Bagaimanakah proses *deployment* sistem informasi ini ?
10	P10	Data apa saja yang disajikan dalam *monitoring cache server* ?
11	P11	Siapakah kategori pengguna dalam sistem informasi ini ?
12	P12	Apakah sistem informasi *monitoring cache server* ini memerlukan fitur *realtime* untuk memperbarui data *cache server* yang tersimpan di database ? jika ya, bagaimana cara implementasinya ?
13	P13	Berapakah batas waktu maksimal respons API yang ingin dikembangkan ?
14	P14	Apakah perusahaan memiliki standar pengkodean untuk aplikasi yang dikembangkan ?
15	P15	Apakah proses *query* pada *database* menggunakan sintaks SQL ?

Setelah melaksanakan proses wawancara kepada narasumber, terdapat jawaban yang telah diterima dari pertanyaan tabel 3.3, sebagai berikut.

Tabel 3.3 Jawaban Pertanyaan Wawancara

No.	Kode	Jawaban
1	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 log *cache proxy* meliputi *access log, store log, useragent log, cache log*
2	P2	Selama ini perusahaan memantau *cache proxy* dengan menganalisis data yang diperoleh dari *SSH* yang terhubung ke server *proxy*.
3	P3	Pernah terjadi kegagalan fungsi *proxy server squid* diakibatkan *overload* pengguna

4	P4	data *access log, store log, user agent log, cache lo, server proxy,* dan data *user*
6	P6	Iya, sistem ini memerlukan autentikasi dalam penggunanya yang bertujuan untuk melindungi sistem informasi dari kebobolan data dan mengetahui siapa saja yang diberikan hak untuk menggunakan aplikasi ini.
7	P7	*Middleware* yang digunakan dalam sistem informasi ini adalah autentikasi token yang berasal dari *JSON Web Token* (*JWT*), *middleware* bawaan *Django* yaitu CORS (*Cross-Origin Resource Sharing*) dan *HTTPS Security*
8	P8	Sistem ini menggunakan *Framework Django* dan *React JS* dalam pengembanganya.
9	P9	*Production* sistem informasi ini berjalan di server perusahaan.
11	P11	Administrator
12	P12	Ya, sistem informasi ini membutuhkan fitur *realtime* dalam proses memperbarui data, dikarenakan perusahaan membutuhkan data aktual terkait *log cache proxy* sebagai bahan analisis untuk menentukan masalah di masa depan.
13	P13	Maksimal waktu respon API yang dibuat adalah 600 ms
14	P14	Ya perusahaan memiliki standar pengkodean agar mempermudah proses pemeliharaan kedepanya
15	P15	Tidak, Proses *query database* yang dilakukan hanya perlu menggunakan model yang telah di sediakan *django* saja

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.*

Setelah melaksanakan serangkaian kegiatan seperti wawancara, observasi, dan studi literatur mengenai topik *monitoring cache* server, berhasil mengumpulkan

sejumlah data. Data yang terkumpul ini berupa data fungsional dan non-fungsional sistem. Dibawah ini Tabel 3.4 dan Tabel 3.5, berisikan kebutuhan fungsional dan non fungsional sistem.

Tabel 3.4 Kebutuhan Fungsional

No.	Kode	Kebutuhan Fungsional
1	KF1	Melakukan koneksi otomatis ke server *proxy* menggunakan SSH untuk mengambil data log *cache proxy* meliputi data *access log*, *store log*, *user agent log*, dan *cache log* dengan fitur *websocket* yang di dukung oleh *framework django* secara *real time*.
2	KF2	Menyediakan *endpoint* API untuk menyajikan data setiap *record* aktivitas *cache proxy* meliputi data *access log*, *store log*, *user agent log*, dan *cache log* .
3	KF3	Menyediakan *endpoint API* untuk proses login pengguna.
4	KF4	Menyediakan *endpoint API* untuk menambah, mengubah, dan menyajikan data *proxy server*.
5	KF5	Menyediakan *endpoint REST API* untuk mengubah, dan menyajikan data *pribadi pengguna*.

Tabel 3. 5 Kebutuhan non fungsional

No.	Kode	Kebutuhan Non Fungsional
1	KNF1	Waktu *response REST API* maksimal 600 ms (*millisecond*)

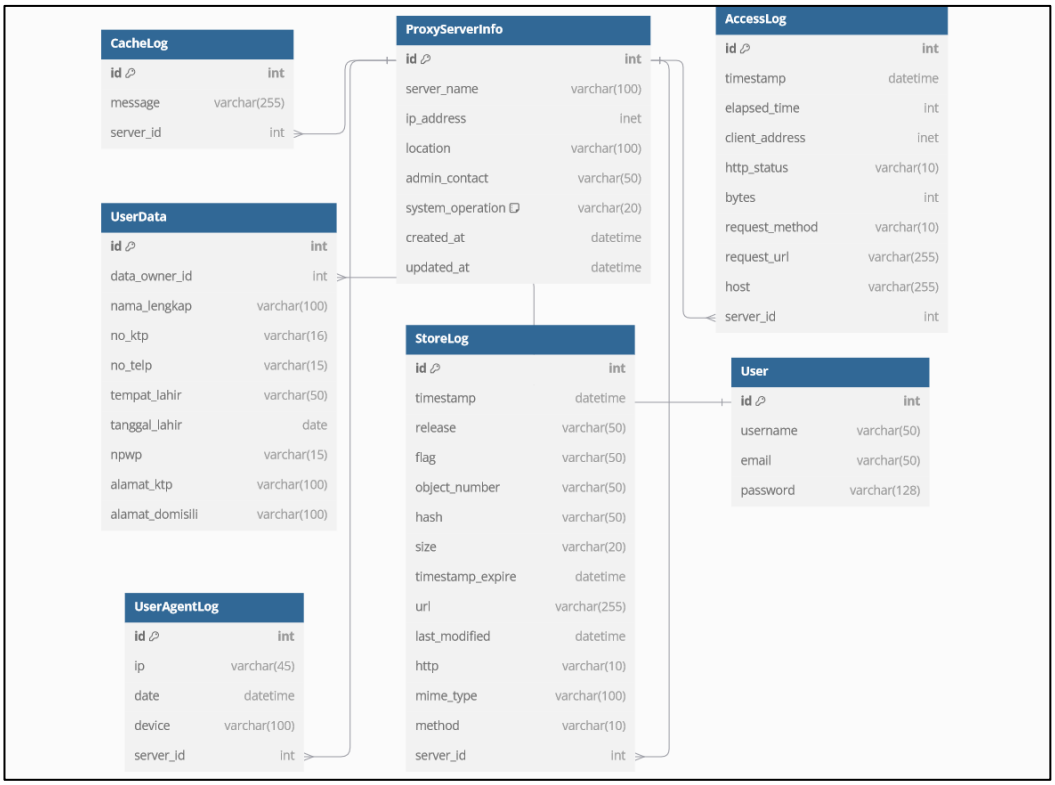
2	KNF2	Otomatisasi SSH harus diimplementasikan dengan *django websocket* agar proses *SSH* bisa dilakukan secara *realtime*
3	KNF3	Penulisan kode harus menggunakan standar pengkodean yang konsisten, agar dapat memberikan kemudahan dalam proses pemeliharaan.
4	KNF4	*Django* harus menggunakan *CORS* (*Cross-Origin Resource Sharing*) diperlukan untuk mengatur kebijakan akses antar domain (*cross-domain*).
5	KNF5	Semua REST API yang memiliki *method* GET harus dilengkapi dengan fitur *search*, *ordering*, *filter* dan *limit offset pagination*
6	KNF6	Membuat model *Django* yang mempresentasikan diagram ERD (Entity Relation Diagram) atau tabel database pada sistem

3.4.2 *Design*

Setelah kebutuhan awal teridentifikasi, langkah selanjutnya adalah melakukan *Design*. *Design* yang diperlukan antara lain *Entity Relationship Diagram* (ERD) dan daftar spesifikasi *end point API*.

3.4.2.1 *Entity Relationship Diagram* (ERD)

Adapun rancangan sistem *database* yang digambarkan pada *entity relationship diagram* sebagai berikut, pada Gambar 3.3.



Gambar 3.3 Diagram ERD sistem *monitoring*

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 berbagai tabel lainnya.

Tabel *UserAgentLog*, *AccessLog*, *CacheLog*, dan *StoreLog* masing-masing berisi data *record* aktivitas *cache proxy* meliputi data *access log, store log, user agent log,* dan *cache log*. Setiap tabel yang memuat data *record* memiliki kolom *server_id*, yang berfungsi sebagai foreign key yang terhubung dengan tabel *ProxyServerInfo*. Ini memungkinkan setiap data *record* aktivitas *cache proxy* terkait dengan server *proxy* tertentu. Adapun penjelasan setiap kolom tabel basis data yang berisikan masing-masing data *record* aktivitas *cache proxy* pada tabel-tabel dibawah ini, meliput Tabel 3.6, Tabel 3.7, Tabel 3.8 dan Tabel 3.9.

Tabel 3.6 Penjelasan kolom pada tabel basis data *CacheLog*

No	Tabel *Cache Log*	
	Nama Kolom	Deskripsi
1	*message*	Pesan log yang berisi informasi tentang operasi *cache Squid*, misalnya status atau peringatan.
2	*server*	Referensi ke tabel *ProxyServerInfo*, yang menyimpan informasi tentang server *Squid* yang mencatat log ini.

Tabel 3.7 Penjelasan kolom pada tabel basis data *StoreLog*

No	Tabel *Store Log*	
	Kolom	Deksirpsi
1	*timestamp*	Waktu ketika *log* dicatat
2	*realese*	Indikator apakah objek dilepas atau tetap disimpan di *cache*.
3	*flag*	*Flag* yang menunjukkan status objek (misalnya status validitas atau status khusus lain yang digunakan *Squid*).
4	*object_number*	Nomor unik objek yang dicatat dalam *cache Squid*, digunakan untuk identifikasi objek.
5	*hash*	*Hash* unik objek, berfungsi sebagai tindakan *log*.
6	*size*	Ukuran objek yang dicatat dalam *log cache Squid*.
7	*timestamp_expire*	Waktu kedaluwarsa dari objek di *cache*, menandakan kapan objek tidak valid lagi.
8	*url*	URL dari objek yang disimpan atau dilepas oleh *Squid Proxy*.
9	*last_modified*	Waktu modifikasi terakhir dari objek, yang menunjukkan kapan objek terakhir kali diperbarui.
10	*http*	Status HTTP objek (misalnya 200, 404)

11	*mime_type*	Tipe MIME dari objek, seperti *text/*html atau *image*/png, digunakan untuk menunjukkan tipe konten objek.
12	*methode*	Metode HTTP yang digunakan (misalnya GET, POST).
13	*server*	Referensi ke tabel *ProxyServerInfo* yang berisi informasi tentang server Squid yang mencatat log ini.

Tabel 3.8 Penjelasan kolom pada tabel basis data *UserAgentLog*

No	Tabel *User Agent Log*	
	Nama Kolom	Deskripsi
1	*ip*	Alamat IP dari perangkat yang melakukan permintaan HTTP.
2	*date*	Waktu atau tanggal permintaan HTTP dibuat.
3	*device*	Informasi perangkat yang terhubung, biasanya mengindikasikan tipe perangkat atau User-Agent yang digunakan.
4	server	Referensi ke tabel ProxyServerInfo yang berisi informasi tentang server Squid yang mencatat log ini.

Tabel 3.9 Tabel penjelasan kolom pada tabel basis data *AccessLog*

Tabel Access Log	
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.

Di sisi pengguna, tabel UserData menyimpan informasi pribadi pengguna, termasuk nama lengkap, nomor KTP, jenis kelamin, nomor telepon, tempat dan tanggal lahir, NPWP, agama, alamat KTP, serta alamat domisili. Tabel *UserData* memiliki kolom *data_owner_id*, yang merupakan *foreign key* yang terhubung ke tabel *User*, di mana detail pengguna seperti *username*, *email*, dan *password* disimpan. Relasi ini memungkinkan *UserData* mengakses data pemilik melalui satu entitas *User*.

3.4.2.2 *Endpoint* API

Pada proses desain ini dikembangkan juga beberapa *endpoint* API untuk mendukung kebutuhan sistem *monitoring*, karena terdapat dua iterasi selama proses pengembangan maka didapatkan penambahan *endpoint* API pada iterasi kedua. Berikut *endpoint* API pada kedua iterasi.

a. Endpoint API pada Iterasi Satu

Adapun *endpoint* API pada iterasi satu sebagai berikut.

1) *Endpoint* API untuk *Login*

Adapun Tabel 3.10 deskripsi *endpoint* API untuk *sign in* pada tabel dibawah ini.

Tabel 3.10 Tabel spesifikasi *endpoint* API untuk *Login*

Spesifikasi *endpoint* API untuk *Login*	
Method	Post
Endpoint	http://host:8000/auth
Deskripsi	Digunakan Untuk Login Penggguna
Izin	Terbuka
Params	-
Body (JSON)	Username (string) dan Password (string)
Header	-
Response	JSON
Status Code (*OK*)	200 Ok
Status Code (*Bad Request*)	400 *Bad Request*
Status Code (*Unauthorized*)	401 *Unauthorized*

Tabel di atas adalah rancangan *endpoint login* menggunakan *Django REST Framework* (DRF) untuk memungkinkan pengguna melakukan *login* dengan *username* dan *password* serta mendapatkan token untuk otorisasi, dengan URL *Endpoint* `http/host:8000/auth`. API ini memvalidasi kredensial pengguna, lalu menghasilkan token JWT (*refresh* dan *access*). Jika validasi berhasil, respons berisi detail pengguna seperti ID pengguna, kunci token, serta token JWT yang berformat JSON dan akan dikembalikan dengan status HTTP 200. Namun, jika autentikasi gagal, respons berupa pesan kesalahan dengan status HTTP 401 dikembalikan. Adapun dari response JSON dihasilkan dari API diatas jika pengguna berhasil login dibawah ini. Di bawah ini Gambar 3.4 menggambar kan r*esponse* JSON dari API ini.

```
{
    "user_id": 1,
    "key": "982f1a8b9d96738e67bceb20fdad3ee5f5c9284c",
    "jwt_refresh": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
        eyJ0b2tlbl90eXBlIjoicmVmcmVzaCIsImV4cCI6MTczNjI3NDE5OCwiaWF0IjoxNzM2MTg3Nzk4LCJqdGkiOiI5Yjk1NGIwMD
        BjYzc0Mzg2YTBhZmUxOTI0MDRiOTUxNiIsInVzZXJfaWQiOjF9.c05A68UTLGMB2uqoSs5kGvSBQO9DHh0dU8lFQzWLuF4",
    "jwt_access": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
        eyJ0b2tlbl90eXBlIjoiYWNjZXNzIiwiZXhwIjoxNzM2Mjc0MTk4LCJpYXQiOjE3MzYxODc3OTgsImp0aSI6ImZiOWMzYTBkMz
        RkYzQzOWNhNWNiMWI0ZjBkMjA3YmU5IiwidXNlcl9pZCI6MX0.q6xiSO-rcDaslxhUC4ppaDONuMwsgfny_pCn1WIiBKs",
    "status": 200,
    "message": true
}
```


Gambar 3.4 Response JSON *enpoint* API untuk *login* pengguna

2) *Endpoint* API untuk Memperbarui Data Pengguna

Adapun Tabel 3.11 deskripsi endpoint API untuk edit data pengguna pada tabel dibawah ini.

Tabel 3.11 Deskripsi *endpoint* API untuk memperbarui data pengguna

Spesifikasi *endpoint* API untuk *Login*	
Method	Put
Endpoint	http://host:8000/update_data_user_id/<int:id>/
Deskripsi	Digunakan untuk mengedit data pengguna
Izin	Perlu izin token autentikasi menggunakan *Java Script Web Token* (JWT)
Params	Id pengguna (*integer*)
Body (JSON)	Username (String), Password (String), Nama Lengkap(String), Jenis Kelamin(String),No KTP (String), Alamat Domisili(String), Alamat KTP (String), NPWP(Number), No Telp(String), Agama(String), Tanggal Lahir (Date), Tempat Lahir (Date)
Header	Token *Java Script Web Token* (JWT)
Response (JSON)	Message (String)
Status Code (*OK*)	200 Ok
Status Code (*Bad Request*)	400 *Bad Request*

Status Code (*Unauthorized*)	401 *Unauthorized*

Tabel diatas adalah endpoint API untuk memperbarui data pengguna berdasarkan ID yang diterima melalui URL, dengan URL *endpoint* `host:8000/update_data_user_id/<int:id>/`. API ini memiliki metode PUT, serta dilindungi oleh izin autentikasi. Ketika metode PUT digunakan, fungsi ini mengambil data dari permintaan pengguna dan memastikan data tersebut valid. Kemudian, kode mencoba mendapatkan objek pengguna berdasarkan ID yang diberikan. Jika pengguna ditemukan, atribut-atribut seperti *username*, *password*, dan email. Selain itu, fungsi ini juga mengelola data tambahan pengguna yang tersimpan dalam model *UserData*. Jika data tambahan sudah ada, atribut-atributnya diperbarui berdasarkan data permintaan, sedangkan jika tidak ada, data baru akan dibuat menggunakan data yang disediakan. Seluruh perubahan disimpan, dan respons dengan status HTTP 200 akan diberikan jika pembaruan berhasil. Jika terjadi kesalahan, seperti ID pengguna yang tidak ditemukan atau data yang tidak valid, fungsi ini akan mengembalikan respons kesalahan dengan pesan yang sesuai. *Endpoint* ini dirancang untuk memberikan mekanisme yang aman dan fleksibel dalam memperbarui informasi pengguna serta data terkait di sistem.

3) *Endpoint* API untuk Mengambil Seluruh Data *Cache Log*

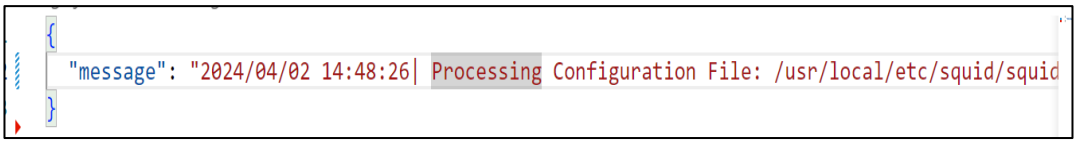
Adapun Tabel 3.12 deskripsi *endpoint* API untuk mengambil seluruh data *cache log* pada tabel dibawah ini.

Tabel 3.12 Deskripsi *endpoint* API untuk mengambil seluruh data *cache log*

Spesifikasi *endpoint* API untuk Mengambil Seluruh Data Cache Log	
Method	Get
Endpoint	http://host:8000/api/cachelogview/
Deskripsi	Digunakan untuk mengambil semua data *cache log*
Izin	Perlu izin token autentikasi menggunakan *Java Script Web Token* (JWT)

Params	-
Body (JSON)	-
Header	Token *Java Script Web Token* (JWT)
Response	JSON
Status Code (*OK*)	200 Ok
Status Code (*Bad Request*)	400 *Bad Request*
Status Code (*Unauthorized*)	401 *Unauthorized*

Tabel di atas adalah implementasi *endpoint* API untuk melihat atau membuat log akses yang terhubung ke model *CacheLog*, dengan URL endpoint `http://host:8000/acceslogview/`, yang memungkinkan pengguna untuk melakukan operasi GET untuk mengambil daftar data *cache log*. Autentikasi menggunakan *TokenAuthentication*, sementara izin akses diatur menggunakan *IsAuthenticated* sehingga hanya pengguna yang telah terautentikasi yang dapat mengakses *endpoint* ini. *Endpoint* ini juga mendukung paginasi menggunakan *limit offset pagination* serta fitur filter dan pencarian data dengan memanfaatkan *search* dan *ordering*. Dengan filter ini, pengguna dapat memfilter data berdasarkan kolom seperti id, *http_status*, dan *request_url*, serta melakukan pencarian berdasarkan kolom yang sama. Adapun dari response JSON dihasilkan dari API diatas. Di bawah ini Gambar 3.5 menggambar kan r*esponse* JSON dari API ini.



Gambar 3.5 *Response* JSON *enpoint* API untuk mengambil seluruh data *cache*

4) *Endpoint* API untuk Mengambil Seluruh Data *Store Log*

Adapun Tabel 3.13 deskripsi *endpoint* API untuk mengambil seluruh data *store log* pada tabel dibawah ini.

Tabel 3.13 Deskripsi *endpoint* API untuk mengambil seluruh data *store log*

Spesifikasi *endpoint* API untuk Mengambil Seluruh Data *Store Log*	
Method	Get
Endpoint	http://host:8000/api/storelogview/
Deskripsi	Digunakan untuk mengambil semua data store *log*
Izin	Perlu izin token autentikasi menggunakan *Java Script Web Token* (JWT)
Params	-
Body (JSON)	-
Header	Token *Java Script Web Token* (JWT)
Response	JSON
Status Code (*OK*)	200 Ok
Status Code (*Bad Request*)	400 *Bad Request*
Status Code (*Unauthorized*)	401 *Unauthorized*

Tabel di atas adalah implementasi *endpoint* API untuk melihat data *store log* yang terhubung ke model StoreLog, dengan URL *endpoint* `http://host:8000/storelogview/`, yang memungkinkan pengguna untuk melakukan operasi GET untuk mengambil daftar data *store log*. Autentikasi menggunakan *TokenAuthentication*, sementara izin akses diatur menggunakan *IsAuthenticated* sehingga hanya pengguna yang telah terautentikasi yang dapat mengakses *endpoint* ini. *Endpoint* ini juga mendukung paginasi menggunakan *limit offset pagination* serta fitur filter dan pencarian data dengan memanfaatkan *search* dan *ordering*. Dengan filter ini, pengguna dapat memfilter data berdasarkan kolom database, serta melakukan pencarian berdasarkan kolom yang sama. *Endpoint* ini dirancang untuk

memberikan kemudahan untuk membaca data secara terstruktur. Di bawah ini Gambar 3.6 menggambar kan r*esponse* JSON dari API ini.

```
{
  "timestamp": "1722911319.819",
  "realese": "RELEASE",
  "flag": "-1",
  "object_number": "FFFFFFFF",
  "hash": "02000000000000001037000001000000",
  "size": "22/22",
  "timestamp_expire": "-1",
  "url": "http://www.msftconnecttest.com/connecttest.txt?",
  "last_modified": "1722911960",
  "http": "200",
  "mime_type": "text/plain",
  "methode": "GET"
},
```


Gambar 3.6 *Response* JSON *enpoint* API store log berdasarkan *proxy server*

5) *Endpoint* API untuk Mengambil Seluruh Data *User Agent Log*

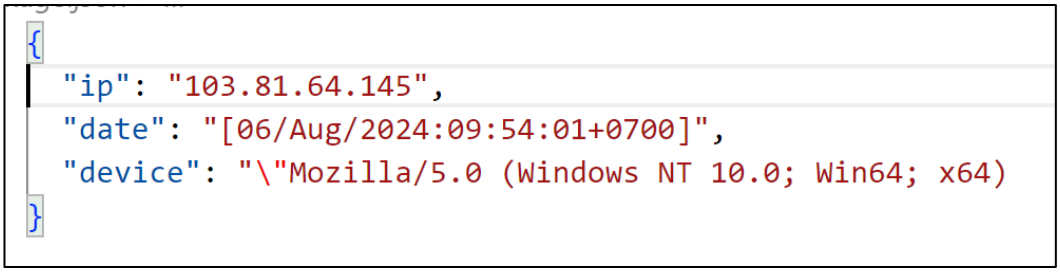
Adapun Tabel 3.14 deskripsi *endpoint* API untuk mengambil seluruh data *user agent log* pada tabel dibawah ini.

Tabel 3.14 Tabel deskripsi *endpoint* API untuk mengambil seluruh data *user agent log*

Spesifikasi *endpoint* API untuk Mengambil Seluruh Data *User Agent Log*	
Method	Get
Endpoint	http://host:8000/api/agentlogview/
Deskripsi	Digunakan untuk mengambil semua data *user agent log*
Izin	Perlu izin token autentikasi menggunakan *Java Script Web Token* (JWT)
Params	-
Body (JSON)	-

Header	Token *Java Script Web Token* (JWT)
Response	JSON
Status Code (*OK*)	200 Ok
Status Code (*Bad Request*)	400 *Bad Request*
Status Code (*Unauthorized*)	401 *Unauthorized*

Tabel di atas adalah implementasi *endpoint* API untuk melihat data *user agent log* yang terhubung ke model *UserAgentLog*, dengan URL *endpoint* `http://host:8000/agentlogview/`, yang memungkinkan pengguna untuk melakukan operasi GET untuk mengambil data *user agent log*. Autentikasi menggunakan *TokenAuthentication*, sementara izin akses diatur menggunakan *IsAuthenticated* sehingga hanya pengguna yang telah terautentikasi yang dapat mengakses *endpoint* ini. *Endpoint* ini juga mendukung paginasi menggunakan *limit offset pagination* serta fitur filter dan pencarian data dengan memanfaatkan *search* dan *ordering*. Dengan filter ini, pengguna dapat memfilter data berdasarkan model, serta melakukan pencarian berdasarkan kolom yang sama. *Endpoint* ini dirancang untuk memberikan kemudahan untuk membaca data secara terstruktur. Di bawah ini Gambar 3.7 menggambar kan r*esponse* JSON dari API ini.



```
{
  "ip": "103.81.64.145",
  "date": "[06/Aug/2024:09:54:01+0700]",
  "device": "\"Mozilla/5.0 (Windows NT 10.0; Win64; x64)
}
```


Gambar 3.7 *Response* JSON *enpoint* API *user agent log* berdasarkan *proxy server*

6) *Endpoint* API untuk Mengambil Seluruh Data *Access Log*

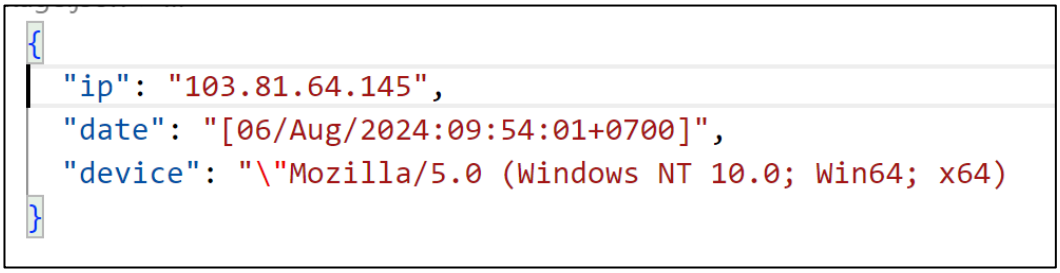
Adapun Tabel 3.15 deskripsi *endpoint* API untuk mengambil seluruh data *access log* pada tabel dibawah ini.

Tabel 3.15 Deskripsi *endpoint* API untuk mengambil seluruh data *access log*

Spesifikasi *endpoint* API untuk Mengambil Seluruh Data *Acces Log*	
Method	Get
Endpoint	http://host:8000/api/accesslogview/
Deskripsi	Digunakan untuk mengambil semua data *access log*
Izin	Perlu izin token autentikasi menggunakan *Java Script Web Token* (JWT)
Params	-
Body (JSON)	-
Header	Token *Java Script Web Token* (JWT)
Response	JSON
Status Code (*OK*)	200 Ok
Status Code (*Bad Request*)	400 *Bad Request*
Status Code (*Unauthorized*)	401 *Unauthorized*

Tabel di atas adalah implementasi *endpoint* API untuk melihat data *user agent log* yang terhubung ke model *AccessLog*, dengan URL *endpoint* `http://host:8000/accesslogview/`, yang memungkinkan pengguna untuk melakukan operasi GET untuk mengambil data *access log*. Autentikasi menggunakan *TokenAuthentication*, sementara izin akses diatur menggunakan *IsAuthenticated* sehingga hanya pengguna yang telah terautentikasi yang dapat mengakses *endpoint* ini. *Endpoint* ini juga mendukung paginasi menggunakan *limit offset pagination* serta fitur filter dan pencarian data dengan memanfaatkan *search* dan *ordering*. Dengan filter ini, pengguna dapat memfilter data berdasarkan model, serta melakukan pencarian berdasarkan kolom yang sama. *Endpoint* ini dirancang

untuk memberikan kemudahan untuk membaca data secara terstruktur. Di bawah ini Gambar 3.8 menggambar kan r*esponse* JSON dari API ini.



Gambar 3.8 Response JSON *enpoint* API user agent *log* berdasarkan *proxy server*

***7) Endpoint* API untuk Mengambil dan Menambah Data Server Proxy**

Adapun Tabel 3.16 deskripsi *endpoint* API untuk menambah data server *proxy* pada tabel dibawah ini.

Tabel 3.16 Tabel deskripsi *endpoint* API untuk menambah data server *proxy*

Spesifikasi *endpoint* API untuk Menambah Data Server *Proxy*	
Method	Post dan Get
Endpoint	http://host:8000/servercreate/
Deskripsi	Digunakan untuk mengambil semua data server proxy dan menambahkan data server *proxy*
Izin	Perlu izin token autentikasi menggunakan *Java Script Web Token* (JWT)
Params	-
Body (JSON)	Nama Server Proxy (String), IP Address (String), Username (String), Password (String), Location (String), Admin Contact (String), Sistem Operasi (String), Created (Date), Updated(Date)
Header	Token *Java Script Web Token* (JWT)

Response	JSON
Status Code (*OK*)	200 Ok
Status Code (*Bad Request*)	400 *Bad Request*
Status Code (*Unauthorized*)	401 *Unauthorized*

API di atas adalah sebuah *endpoint* untuk menangani operasi GET dan POST pada model *ProxyServerInfo* menggunakan *Django REST Framework*. Endpoint tersebut terhubung dengan URL path `http://host:8000/createserver/`. Ketika GET diakses, API akan menampilkan daftar data dari model ProxyServerInfo, dilengkapi fitur pencarian, pengurutan, dan filter. Data juga dapat diurutkan atau difilter berdasarkan atribut tertentu yang telah didefinisikan. Ketika POST diakses, API memungkinkan pengguna untuk menambahkan data baru ke model *ProxyServerInfo*. Dalam metode *create*, jika data yang dikirimkan valid, data akan disimpan ke database dan API mengembalikan respon sukses dengan status 200 OK. Namun, jika data tidak valid, API akan mengembalikan pesan kegagalan dengan status 400 *Bad Request*.

Autentikasi menggunakan *JWTAuthentication*, tetapi akses diizinkan untuk semua pengguna karena permission-nya adalah *IsAuthenticated*. Data yang dikembalikan memiliki *pagination* menggunakan *LimitOffsetPagination*. Di bawah ini Gambar 3.9 menggambar kan r*esponse* JSON dari API ini.


```
{
  "id": 1,
  "timestamp": "1712044454.510",
  "elapsed_time": 18,
  "client_address": "103.81.64.186",
  "http_status": "TCP_DENIED/403",
  "bytes": 3843,
  "request_method": "GET",
  "request_url": "http://www.msftconnecttest.com/connecttest.txt",
  "host": "HIER_NONE/-",
  "server": 1
}
```


Gambar 3.9 Response JSON *enpoint* API *access log* berdasarkan *proxy server*

8) ***Endpoint* API untuk Memperbarui Data Server *Proxy***

Adapun Tabel 3.17 deskripsi *endpoint* API untuk memperbarui data server *proxy* pada tabel dibawah ini.

Tabel 3.17 Tabel deskripsi *endpoint* API untuk memperbarui data *server proxy*

Spesifikasi *endpoint* API untuk Mengubah Data Server *Proxy*	
Method	PUT
Endpoint	http://host:8000/serverupdate/<int:id>/
Deskripsi	Digunakan untuk memperbarui data server *proxy*
Izin	Perlu izin token autentikasi menggunakan *Java Script Web Token* (JWT)
Params	Id server proxy (*integer*)
Body (JSON)	Nama Server Proxy (String), IP Address (String), Username (String), Password (String), Location (String), Admin Contact (String), Sistem Operasi (String), Created (Date), Updated (Date)
Header	Token *Java Script Web Token* (JWT)
Response (JSON)	JSON
Status Code (*OK*)	200 Ok
Status Code (*Bad Request*)	400 *Bad Request*
Status Code (*Unauthorized*)	401 *Unauthorized*

API di atas adalah endpoint yang berfungsi untuk mengambil dan memperbarui data berdasarkan ID Server dari model *ProxyServerInfo*. Endpoint ini diakses melalui URL dengan pola `http://host:8000/updateserver/<int:id>`, di mana `<int:id>` adalah ID Server dari data yang ingin diakses atau diubah. API ini secara default mendukung metode GET untuk mengambil data dan PUT/PATCH untuk memperbarui data. Setelah data berhasil diubah, API akan mengembalikan respon

dengan status 200 OK dan pesan "Data Berhasil Di Ubah". Autentikasi dilakukan menggunakan TokenAuthentication, dan hanya pengguna yang telah terautentikasi (*IsAuthenticated*) yang diizinkan untuk mengakses *endpoint* ini.

b. Endpoint API pada Iterasi Dua

Adapun *endpoint* API pada iterasi dua sebagai berikut.

1) Endpoint API untuk Mengambil Data *Cache Log* Berdasarkan *Proxy Server*

Adapun Tabel 3.18 deskripsi *endpoint* API untuk mengambil data *cache log* berdasarkan *proxy server*.

Tabel 3.18 Tabel deskripsi *endpoint* API untuk mengambil data *cache log*

Spesifikasi *endpoint* API untuk Mengambil Seluruh Data *cache log* berdasarkan server *proxy*	
Method	Post
Endpoint	http://host:8000/api/cachelogviewfilter/
Deskripsi	Digunakan untuk mengambil semua data *cache log* berdasarkan server *proxy*
Izin	*IsAuthenticated* yaitu akses memerlukan token autentikasi menggunakan *Java Script Web Token* (JWT)
Params	-
Body (JSON)	Server Id (Integer)
Header	Token *Java Script Web Token* (JWT)
Response	JSON
Status Code (*OK*)	200 Ok
Status Code (*Bad Request*)	400 *Bad Request*
Status Code (*Unauthorized*)	401 *Unauthorized*

API di atas adalah *endpoint* yang menangani permintaan untuk menampilkan data *cache log* berdasarkan relasi ID Server yang ada pada tabel database *ProxyServerInfo*. *Endpoint* ini diakses melalui URL `http://host:8000/updateserver/cachelogviewfilter/` menggunakan metode POST. Setelah data didapatkan, hasil tersebut dipaginasi dengan aturan yang ditentukan oleh *CustomLimitOffsetPagination*, seperti jumlah *default* item per halaman dan batas maksimum. Data hasil paginasi kemudian diubah menjadi format JSON dengan atribut tertentu seperti waktu akses, alamat klien, status HTTP, dan lainnya. API juga mengembalikan jumlah total log akses.

Endpoint ini hanya dapat diakses oleh pengguna yang terautentikasi dengan token (*Token Authentication*) dan memiliki izin yang sesuai (*IsAuthenticated*). Di bawah ini Gambar 3.10 menggambar kan r*esponse* JSON dari API ini.

Gambar 3.10 *Response* JSON *enpoint* API cache *log* berdasarkan *proxy server*

2) Endpoint API untuk Mengambil Data *Store Log* Berdasarkan *Proxy Server*

Adapun Tabel 3.19 deskripsi *endpoint* API untuk mengambil data *store log* berdasarkan *proxy server*.

Tabel 3.19 Tabel deskripsi *endpoint* API untuk mengambil data *store log*

Spesifikasi *endpoint* API untuk Mengambil Seluruh Data *store log* berdasarkan server *proxy*	
Method	Post
Endpoint	http://host:8000/api/storelogviewfilter/
Deskripsi	Digunakan untuk mengambil semua data *store log* berdasarkan server *proxy*

Izin	*IsAuthenticated* yaitu akses memerlukan token autentikasi menggunakan *Java Script Web Token* (JWT)
Params	-
Body (JSON)	Server Id (Integer)
Header	Token *Java Script Web Token* (JWT)
Response	JSON
Status Code (*OK*)	200 Ok
Status Code (*Bad Request*)	400 *Bad Request*
Status Code (*Unauthorized*)	401 *Unauthorized*

API di atas adalah *endpoint* yang menangani permintaan untuk menampilkan data *store log* berdasarkan relasi ID Server yang ada pada tabel database *ProxyServerInfo*. *Endpoint* ini diakses melalui URL `http://host:8000/updateserver/storelogviewfilter/` menggunakan metode POST.

Setelah data didapatkan, hasil tersebut dipaginasi dengan aturan yang ditentukan oleh *CustomLimitOffsetPagination*, seperti jumlah *default* item per halaman dan batas maksimum. Data hasil paginasi kemudian diubah menjadi format JSON dengan atribut tertentu seperti waktu akses, alamat klien, status HTTP, dan lainnya. API juga mengembalikan jumlah total log akses.

Endpoint ini hanya dapat diakses oleh pengguna yang terautentikasi dengan token (*Token Authentication*) dan memiliki izin yang sesuai (*IsAuthenticated*). Di bawah ini Gambar 3.11 menggambar kan r*esponse* JSON dari API ini.

```
{
  "timestamp": "1722911319.819",
  "realese": "RELEASE",
  "flag": "-1",
  "object_number": "FFFFFFFF",
  "hash": "02000000000000001037000001000000",
  "size": "22/22",
  "timestamp_expire": "-1",
  "url": "http://www.msftconnecttest.com/connecttest.txt?",
  "last_modified": "1722911960",
  "http": "200",
  "mime_type": "text/plain",
  "methode": "GET"
},
```


Gambar 3.11 *Response* JSON *enpoint* API store *log* berdasarkan *proxy server*

3) Endpoint API untuk Mengambil Data *User Agent Log* Berdasarkan *Proxy Server*

Adapun Tabel 3.20 deskripsi *endpoint* API untuk mengambil data *user agent log* berdasarkan *proxy server*.

Tabel 3.20 Tabel deskripsi *endpoint* API untuk mengambil data *user agent log*

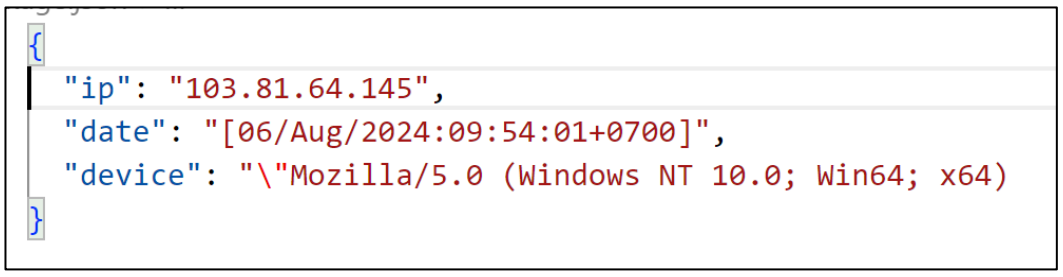
Spesifikasi *endpoint* API untuk Mengambil Seluruh Data *user agent log* berdasarkan server *proxy*	
Method	Post
Endpoint	http://host:8000/api/agentlogviewfilter/
Deskripsi	Digunakan untuk mengambil semua data *user agent log* berdasarkan server *proxy*
Izin	*IsAuthenticated* yaitu akses memerlukan token autentikasi menggunakan *Java Script Web Token* (JWT)
Params	-
Body (JSON)	Server Id (Integer)

Header	Token *Java Script Web Token* (JWT)
Response	JSON
Status Code (*OK*)	200 Ok
Status Code (*Bad Request*)	400 *Bad Request*
Status Code (*Unauthorized*)	401 *Unauthorized*

API di atas adalah *endpoint* yang menangani permintaan untuk menampilkan data *user agent log* berdasarkan relasi ID Server yang ada pada tabel database *ProxyServerInfo*. *Endpoint* ini diakses melalui URL `http://host:8000/updateserver/agentlogviewfilter/` menggunakan metode POST.

Setelah data didapatkan, hasil tersebut dipaginasi dengan aturan yang ditentukan oleh *CustomLimitOffsetPagination*, seperti jumlah *default* item per halaman dan batas maksimum. Data hasil paginasi kemudian diubah menjadi format JSON dengan atribut tertentu seperti waktu akses, alamat klien, status HTTP, dan lainnya. API juga mengembalikan jumlah total log akses.

Endpoint ini hanya dapat diakses oleh pengguna yang terautentikasi dengan token (*Token Authentication*) dan memiliki izin yang sesuai (*IsAuthenticated*). Di bawah ini Gambar 3.12 menggambar kan r*esponse* JSON dari API ini.



Gambar 3.12 *Response* JSON *enpoint* API user agent *log* berdasarkan *proxy server*

4) *Endpoint* API untuk Mengambil Data *Access Log* Berdasarkan *Proxy Server*

Adapun Tabel 3.21 deskripsi *endpoint* API untuk mengambil data *access log* berdasarkan *proxy server*.

Tabel 3.21 Tabel deskripsi *endpoint* API untuk mengambil data *access log*

Spesifikasi *endpoint* API untuk Mengambil Seluruh Data *access log* berdasarkan server *proxy*	
Method	Post
Endpoint	http://host:8000/api/accesslogviewfilter/
Deskripsi	Digunakan untuk mengambil semua data *access log* berdasarkan server *proxy*
Izin	*IsAuthenticated* yaitu akses memerlukan token autentikasi menggunakan *Java Script Web Token* (JWT)
Params	-
Body (JSON)	Server Id (Integer)
Header	Token *Java Script Web Token* (JWT)
Response	JSON
Status Code (*OK*)	200 Ok
Status Code (*Bad Request*)	400 *Bad Request*
Status Code (*Unauthorized*)	401 *Unauthorized*

API di atas adalah *endpoint* yang menangani permintaan untuk menampilkan data *store log* berdasarkan relasi ID Server yang ada pada tabel database *ProxyServerInfo*. *Endpoint* ini diakses melalui URL `http://host:8000/updateserver/storelogviewfilter/` menggunakan metode POST.

Setelah data didapatkan, hasil tersebut dipaginasi dengan aturan yang ditentukan oleh *CustomLimitOffsetPagination*, seperti jumlah *default* item per halaman dan

batas maksimum. Data hasil paginasi kemudian diubah menjadi format JSON dengan atribut tertentu seperti waktu akses, alamat klien, status HTTP, dan lainnya. API juga mengembalikan jumlah total log akses.

Endpoint ini hanya dapat diakses oleh pengguna yang terautentikasi dengan token (*TokenAuthentication*) dan memiliki izin yang sesuai (*IsAuthenticated*). Di bawah ini Gambar 3.13 menggambar kan *response* JSON dari API ini.

Gambar 3.13 Response JSON *enpoint* API *access log* berdasarkan *proxy server*

3.4.2.3 Rancangan Otomatisasi SSH yang Diintegrasikan Dengan *Websocket*

Adapun Tabel 3.22 rancangan Otomatisasi SSH yang Diintegrasikan Dengan *Websocket* sebagai berikut dibawah ini.

Tabel 3.22 Otomatiasi SSH yang Diintegrasikan Dengan Websocket

Komponen	Deskripsi
Endpoint	ws://127.0.0.1:8000/ws/update-user-agent-log/
Class	*UpdateDataLogProxy*
Library/Tools	*channels*, *asyncio*, *paramiko*, *sync_to_async*

Fungsi Utama	Melakukan pembaruan periodik terhadap data log akses (*access log*) melalui koneksi *WebSocket*.
Mekanisme Koneksi	Saat terhubung, *WebSocket* akan menerima koneksi.
	Melakukan pembaruan data setiap interval waktu tertentu.
Protokol Koneksi	*WebSocket Protocol*
Autentikasi SSH	Menggunakan *paramiko* untuk menghubungkan ke server *Squid*.
	Autentikasi menggunakan *hostname*, *username*, *password*, dan *port* (22)
Path Log Squid	/var/log/squid/access.log
Operasi Utama	1. Membaca data dari *Squid log* menggunakan SSH.
	2. Mem-*parsing* data log dan menyimpannya ke database.
Interval Waktu	10 detik (dapat disesuaikan dengan asyncio.sleep).
Pesan *WebSocket*	Sukses: {"*message*": "*Periodic update access completed*"}
	Gagal: {"message": "*Periodic update failed*: <*error_message*>"}

Error Handling	*asyncio.CancelledError: Saat task dihentikan secara paksa.*
	SSHException: Saat terjadi kesalahan koneksi SSH.
Operasi *Database*	Menyimpan *log* baru ke dalam model *AccessLog*.
	Memastikan data tidak duplikat dengan perbandingan jumlah *log* di *cache* dan *database*.
Fungsi Tambahan	*update_periodic_data*(): Fungsi sinkron untuk membaca *log* dan memperbarui data.
Keamanan	Menangani kunci host SSH menggunakan *AutoAddPolicy* (dapat ditingkatkan untuk keamanan lebih baik).
Manajemen Koneksi	Koneksi SSH ditutup dengan *parami.close*() pada bagian *finally*.

3.4.3 *Constructor*

Tahap ketiga adalah *Construction* atau konstruksi. Pada tahap ini dilakukanya pengembangan sistem *monitoring proxy server* berdasarkan kebutuhan sistem dan desain yang telah di buat sebelumnya. Pada tahap ini dilakukan dua metode yaitu pengkodean dan pengujian testing.

3.4.3.1 Pengkodean

Pada tahapan pengkodean ini, terdapat dua iterasi selama proses pengembangan sebagai berikut.

a. Pengkodean Iterasi Satu

Pada tahapan pengkodean iterasi satu, melakukan serangkain pengkodean untuk membangun fitur-fitur yang telah di gambarkan pada kebutuhan fungsional dan design.

b. Pengkodean Iterasi Dua

Pada tahap pengkodean iterasi kedua, dilakukan penambahan *endpoint* API yang tercantum dalam tabel *endpoint* untuk iterasi dua. *Endpoint* yang ditambahkan memiliki fungsi untuk mendapatkan setiap data log meliputi *cache log*, *store log*, *user agent log*, dan *access log* berdasarkan proxy server.

3.4.3.2 *Testing*

Pada penelitian ini dilakukan testing pada dua iterasi.

***a. Testing* Iterasi Satu**

Pada penelitian ini dilakukan proses testing menggunakan metode *black box testing*. Testing dilakukan untuk mengukur keberhasilan *endpoint* API dan otomisasi SSH pada iterasi satu. Berikut sekenario testing pada iterasi satu.

1) Sekenario *Testing Endpoint* API untuk *Login* Pengguna

Sekenario yang dibuat dalam tahapan ini disusun sesuai dengan spesifikasi *endpoint* API yang tercantum pada tabel 3.23 *Endpoint* API untuk login pengguna.

Tabel 3.23 Sekenario *Black Box Testing API Login* Pengguna

Test Case ID	Skenario Pengujian	*Header*	*Body* (JSON)	Hasil yang Diharapkan
POST				
TC1	Autentikasi dengan kredensial valid		*Username*: user1	Status: 200 OK
			Password: password123	Token: <JWT Token>
TC2	Autentikasi dengan username salah		*Username*: invalidUser	Status: 401 Unauthorized
			Password: password123	Pesan: Username Password is required
TC3	Autentikasi dengan password salah		*Username*: user1	Status: 401 Unauthorized
			Password: *wrongPassword*	Pesan: Username Password is required
TC4	Autentikasi tanpa mengisi username		*Username*: ``	Status: 400 Bad Request
			Password: password123	Pesan: Username Password is required
TC5	Autentikasi tanpa mengisi password		*Username*: user1	Status: 400 Bad Request
			Password: ``	Pesan: Username Password is required
TC6	Autentikasi dengan data kosong		*Username*: *Password*:	Status: 400 Bad Request
				Pesan: Username and password are required

2) Sekenario Pengujian *Endpoint* API untuk Memperbarui Data Pengguna

Sekenario yang dibuat dalam tahapan ini disusun sesuai dengan spesifikasi *endpoint* API yang tercantum pada tabel 3.24 Endpoint API untuk memperbarui data pengguna.

Tabel 3.24 Sekenario *Black Box Testing API* Memperbarui Data Pengguna

Test Case ID	Sekenario	*Params*	*Header*	*Body* (JSON)	Hasil Diharapkan
PUT					
TC7	*Update* data pengguna dengan *params* id valid, *authorization* valid dan data lengkap	id : integer (valid)	*Authorization* : *Token* JWT (*Java Script Web Token*)	Data Lengkap	*Code*: 200 *Message*: *Data successfully updated*!
TC8	*Update* data pengguna tanpa *body*	id : integer (valid)	*Authorization* : *Token* JWT (*Java Script Web Token*)	Data Tidak Ada	*Code*: 400 *Message: No data provided*
TC9	*Update* data pengguna dengan ID yang tidak ditemukan	id : integer (tidak valid)	*Authorization* : *Token* JWT *(Java Script Web Token)*	Data Lengkap	*Code: 404* *Message: User not found*
TC10	*Update* data pengguna hanya beberapa *field*	id : integer (valid)	*Authorization* : *Token* JWT *(Java Script Web Token)*	Data Tidak Lengkap	*Code: 400* *Message: No data provided*
TC11	*Update* data pengguna dengan	id : integer (valid)	*Authorization* : *Token* JWT	Data Format Salah	*Code: 400*

	format tanggal salah		*(Java Script Web Token)*		*Message: No data provided*
TC12	*Update* data pengguna dengan *field* yang tidak ada	id : integer (valid)	*Authorization : Token JWT (Java Script Web Token)*	Data Field Kosong	*Code: 400* *Message: No data provided*
TC13	*Update* data pengguna dengan *password kosong*	id : ID yang valid	*Authorization : Token JWT (Java Script Web Token)*	Data Password Kosong	*Code: 200* *Message: Data successfully updated!*
TC15	Update data pengguna tanpa token autentikasi	id : ID yang valid	*Authorization : -*	Data Lengkap	*Code : 401, Message : Unauthorized*

3) Skenario Pengujian *Endpoint* API untuk Mengambil Semua Data Setiap *Log*

Tabel 3.25 berisikan sekenario pengujian yang dibuat untuk spesifikasi *endpoint* API yang tercantum pada tabel 3.10 *Endpoint* API untuk mengambil semua data *chace log*, 3.11 *Endpoint* API untuk mengambil semua data *store log*, 3.12 *Endpoint* API untuk mengambil semua data *user agent log*, dan 3.13 *Endpoint* API untuk mengambil semua data *access log*.

Tabel 3.25 Sekenario *black box testing API untuk* mengambil seluruh data setiap *log*

No.	Test Case	Header	Params	Output yang Diharapkan
Get				

TC16	Mendapatkan semua data log dengan token autentikasi	Authorization : Token JWT (Java Script Web Token)	tidak ada	Status respons 200 OK, mengembalikan data semua *log*.
TC17	Mendapatkan semua data log berdsarkan *filter* yang diberikan. filter berdasarkan kolom pada baris dalam tabel database	Authorization : Token JWT (Java Script Web Token)	menyisipkan parameter field pada url *endpoint* API Contoh : /urlendpoint/?field=1	Status respons 200 OK, mengembalikan daftar log dengan mengandung setiap data *field* yang telah dimasukan.
TC18	Mendapatkan semua data log berdsarkan *search* yang diberikan. Search mencari semua data berdasarkan kolom pada baris dalam tabel database	Authorization : Token JWT (Java Script Web Token)	menyisipkan parameter search pada url *endpoint* API Contoh : /urlendpoint/?search=youtube.com	Status respons 200 OK, mengembalikan semua data setiap log dengan mengandung data *field* yang memiliki nilai yang sama pada parameter search.
TC19	Mendapatkan semua data log berdsarkan paginasi limit dan offset yang diberikan	Authorization : Token JWT (Java Script Web Token)	menyisipkan parameter limit dan offset pada url endpoint API Contoh : /urlendpoint/?limit=10&offset=0	Status respons 200 OK, mengembalikan sejumlah data log berdasarkan limit dan offset.

TC20	Urutkan berdasarkan id secara ascending	Authorization : Token JWT (Java Script Web Token)	menyisipkan parameter ordering ascending pada url endpoind : /api/?ordering=id	Status respons 200 OK, mengembalikan log cache yang diurutkan berdasarkan id secara ascending.
TC21	Urutkan berdasarkan id secara descending	Authorization : Token JWT (Java Script Web Token)	menyisipkan parameter ordering ascending pada url endpoind : /api/?ordering=-id	Status respons 200 OK, mengembalikan log cache yang diurutkan berdasarkan id secara descending.
TC22	Mendapatkan data jika tabel log pada database tidak ada	Authorization : Token JWT (Java Script Web Token)	tidak ada	Status respons 400 Bad Request
TC23	Akses tanpa token autentikasi	Authorization : -	tidak ada	Status respons 401 Unauthorized, mengembalikan pesan error yang menunjukkan bahwa otentikasi diperlukan.

4) Sekenario Pengujian *Endpoint* API untuk Mengambil dan Menambah Data *Proxy Server*

Sekenario yang dibuat dalam tahapan ini disusun sesuai dengan spesifikasi *endpoint* API yang tercantum pada tabel 3.26 *Endpoint* API untuk mengambil dan menambah data server proxy.

Tabel 3.26 Sekenario *black box testing API* untuk mengambil dan menambah data *proxy server*

ID *Test Case*	Skenario	*Heade*r	*Body*(JSON)	Output yang Diharapkan
GET				
TC24	Meminta data server tanpa token otentikasi	Authorization : -	tidak ada	401 Unauthorized (Tidak Diizinkan)
TC25	Meminta data server dengan token JWT yang valid	Authorization : Token JWT (Java Script Web Token)	tidak ada	200 OK, daftar server dalam format JSON
TC26	Meminta data server dengan token JWT yang tidak valid	Authorization : Token JWT (Java Script Web Token)	tidak ada	401 Unauthorized (Tidak Diizinkan)
TC27	Meminta data server dengan token JWT yang sudah kedaluwarsa	Authorization : Token JWT (Java Script Web Token)	tidak ada	401 Unauthorized (Tidak Diizinkan)
TC28	Meminta data server dengan batas paginasi dan offset tertentu	Authorization : Token JWT (Java Script Web Token)	tidak ada	200 OK, daftar server dalam format JSON berdasarkan paginasi
TC29	Meminta data server tanpa parameter paginasi	Authorization : Token JWT (Java Script Web Token)	tidak ada	200 OK, daftar server dengan paginasi *default*

POST				
TC30	Memasukkan data server baru dengan payload yang valid	Authorization : Token JWT (Java Script Web Token)	data server valid	200 OK, pesan Cache Add, data server baru ditambahkan dalam JSON
TC31	Memasukkan data server baru dengan payload yang tidak valid	Authorization : Token JWT (Java Script Web Token)	data server tidak valid	400 Bad Request, pesan Article Created dengan data kesalahan
TC32	Memasukkan data server baru tanpa token otentikasi	Authorization : Token JWT (Java Script Web Token)	data server valid	401 Unauthorized (Tidak Diizinkan)
TC33	Meminta dengan metode HTTP yang tidak valid (mis. PUT)	Authorization : Token JWT (Java Script Web Token)	Metode PUT	405 Method Not Allowed (Metode Tidak Diizinkan)

5) Sekenario Pengujian *Endpoint* API untuk Mengubah Data *Proxy Server*

Sekenario yang dibuat dalam tahapan ini disusun sesuai dengan spesifikasi *endpoint* API yang tercantum pada tabel 3.27 *Enpoint* API untuk mengubah data server *proxy*.

Tabel 3.27 Sekenario *Black Box Testing API* pengubah data *proxy server*

ID Test Case	Skenario	Header	Params	Body(JSON)	Output yang Diharapkan
PUT					
TC34	Memasukkan data server baru dengan payload yang valid	Authorization : Token JWT (Java Script Web Token)	Server ID (Valid)	data server valid	200 OK, pesan Cache Add, data server baru ditambahkan dalam JSON
TC35	Memasukkan data server baru dengan payload yang tidak valid	Authorization : Token JWT (Java Script Web Token)	Server ID (Valid)	data server tidak valid	400 Bad Request, pesan Article Created dengan data kesalahan

TC36	Memasukkan data server baru tanpa token otentikasi	Authorization : -	Server ID (Valid)	data server valid	401 Unauthorized (Tidak Diizinkan)
TC37	Memasukkan data server baru tanpa token params server id yang valid	Authorization : Token JWT (Java Script Web Token)	Server ID (Tidak Valid)	data server valid	400 Bad Request, pesan Article Created dengan data kesalahan
TC38	Meminta dengan metode HTTP yang tidak valid (mis. PUT)	Authorization : Token JWT (Java Script Web Token)	Server ID (Valid)	Metode POST	405 Method Not Allowed (Metode Tidak Diizinkan)

6) **Sekenario Pengujian Otomatisasi SSH pada Server *Proxy* untuk Memperbarui Data *Log Cache Proxy* Meliputi *Cache Log*, *Store Log*, *User Agent Log*, dan *Access Log***

Sekenario yang dibuat dalam tahapan ini disusun bertujuan untuk menguji setiap kondisi otomatisasi SSH pada *proxy server* yang diintegrasikan dengan *websocket framework Django* untuk mendapatkan setiap data log *cache proxy* yaitu *cache log*, *store log*, *user agent log*, dan *access log*. Di bawah ini Tabel 3.28 berisikan sekenario pengujian *black box testing* untuk fitur Otomatisasi SSH pada server *proxy*

Tabel 3.28 Sekenario *Black Box Testing* Otomatisasi SSH pada Server *Proxy*

Test ID	Skenario Pengujian	Langkah Pengujian	Hasil yang Diharapkan
TCW01	Koneksi WebSocket berhasil	- Hubungkan klien ke WebSocket.	- WebSocket terhubung dengan sukses.
		- Pastikan tidak ada error pada connect().	- Pesan sukses diterima di klien (jika ada).

TCW02	Koneksi WebSocket ditutup	- Hubungkan klien ke WebSocket.	- Koneksi terputus tanpa error.
		- Lalu tutup koneksi secara manual.	- Tugas periodik dibatalkan tanpa masalah.
TCW03	Pengiriman pesan periodik berhasil	- Hubungkan klien ke WebSocket.	- WebSocket mengirim pesan "Periodic update access completed" setiap 3 detik (atau interval lainnya).
		- Biarkan WebSocket berjalan selama periode tertentu (contoh: 10 menit).	
TCW04	Kesalahan pada update_periodic_data	- Paksa error pada fungsi update_periodic_data (contoh: server SSH salah).	- Pesan error dikirim ke klien melalui WebSocket dengan format {"message": "Periodic update failed: ...}.
TCW05	Data log diperbarui ke database	- Pastikan ada data baru di log Squid (access.log).	- Data baru dari log Squid (access.log) disimpan ke database.
		- Hubungkan klien ke WebSocket.	- Pesan WebSocket: {"message": "Data valid", "data": {...}} dengan data terakhir yang valid.
TCW06	Log file tidak ditemukan	- Paksa error pada proxy server sehingga koneksi SSH dari *paramiko* ke *proxy server* menjadi error	- Pesan error dikirim ke klien: {"status": "error", "message": "..."}.
			- Tidak ada crash pada WebSocket atau server.
TCW07	WebSocket tetap aktif saat koneksi klien terputus	- Hubungkan klien ke WebSocket.	- WebSocket server menangani dengan benar tanpa error.

		- Putuskan koneksi klien secara tiba-tiba.	- Fungsi disconnect() dipanggil untuk membatalkan tugas periodik.
TCW08	Data yang sudah ada tidak disimpan ulang ke database	- Pastikan database sudah memiliki semua data log sebelumnya.	- Data yang sudah ada di database tidak disimpan ulang.
		- Hubungkan klien ke WebSocket.	- Tidak ada data duplikat di database.
TCW9	Koneksi WebSocket dalam mode beban tinggi	- Hubungkan beberapa klien secara bersamaan (misal: 10 klien).	- WebSocket tetap stabil tanpa error.
			- Semua klien menerima pembaruan periodik sesuai interval.

b. Testing Iterasi Dua

Pada penelitian ini dilakukan proses testing menggunakan metode *black box testing*. Testing dilakukan untuk mengukur keberhasilan *endpoint* API pada iterasi dua.

1) Sekenario Pengujuian *Endpoint* API untuk Mengambil Setiap Data *Log Cache Proxy* Berdasarkan Data *Proxy Server*

Tabel 3.29 berisikan sekenario pengujian yang dibuat spesifikasi *endpoint* API yang tercantum pada tabel 3.18 *Endpoint* API untuk mengambil semua data *chace log*, 3.19 *Endpoint* API untuk mengambil semua data *store log*, 3.20 *Endpoint* API untuk mengambil semua data *user agent log*, dan 3.21 *Endpoint* API untuk mengambil semua data *access log*.

Tabel 3.29 Sekenario *Black Box Testing API* untuk mengambil data *log cacheproxy* berdasarkan data *proxy server*

No.	Test Case	Header	Params	Body (JSON)	Output yang Diharapkan
POST					
TC38	Mendapatkan semua data log dengan token autentikasi	Authorization : Token JWT (Java Script Web Token)	tidak ada	Server Id (Integer)	Status respons 200 OK, mengembalikan data semua *log* .
TC39	Mendapatkan semua data log berdsarkan *filter* yang diberikan. filter berdasarkan kolom pada baris dalam tabel database	Authorization : Token JWT (Java Script Web Token)	menyisipkan parameter field pada url *endpoint* API Contoh : /urlendpoint/?field=1	Server Id (Integer)	Status respons 200 OK, mengembalikan daftar log dengan mengandung setiap data *field* yang telah dimasukan.
TC40	Mendapatkan semua data log berdsarkan *search* yang diberikan. Search mencari semua data berdasarkan kolom pada baris dalam tabel database	Authorization : Token JWT (Java Script Web Token)	menyisipkan parameter search pada url *endpoint* API Contoh : /urlendpoint/?search=youtube.com	Server Id (Integer)	Status respons 200 OK, mengembalikan semua data setiap log dengan mengandung data *field* yang memiliki nilai yang sama pada parameter search.
TC41	Mendapatkan semua data log berdsarkan paginasi limit dan offset yang diberikan	Authorization : Token JWT (Java Script Web Token)	menyisipkan parameter limit dan offset pada url endpoint API Contoh : /urlendpoint/?limit=10&offset=0	Server Id (Integer)	Status respons 200 OK, mengembalikan sejumlah data log berdasarkan limit dan offset.
	Urutkan berdasarkan id secara ascending	Authorization : Token JWT (Java Script Web Token)	menyisipkan parameter ordering ascending pada url endpoind : /api/?ordering=id	Server Id (Integer)	Status respons 200 OK, mengembalikan log cache yang diurutkan
TC42	Urutkan berdasarkan id secara descending	Authorization : Token JWT (Java Script Web Token)	menyisipkan parameter ordering ascending pada url endpoind : /api/?ordering=-id	Server Id (Integer)	Status respons 200 OK, mengembalikan log cache yang diurutkan berdasarkan id secara descending.
TC43	Mendapatkan data jika tabel log pada database tidak ada	Authorization : Token JWT (Java Script Web Token)	tidak ada	Server Id (Integer)	Status respons 400 Bad Request
	Akses tanpa token autentikasi	Authorization : -	tidak ada	Server Id (Integer)	Status respons 401 Unauthorized, mengembalikan pesan error yang menunjukkan bahwa otentikasi diperlukan.

3.4.4 *Cutover*

Pada tahap ini, sistem yang telah dikembangkan diimplementasikan ke dalam lingkungan server persusahaan. Proses ini mencakup pembuatan laporan sistem *monitoring* dan *deployment* sistem ke dalam lingkungan perusahaan.

V. KESIMPULAN DAN SARAN

5.1 Kesimpulan

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

1. Pengembangan REST API dan otomatisasi SSH server *proxy* dengan *Django* berhasil diselesaikan menggunakan metode RAD, dengan tingkat keberhasilan pengujian sebesar 92%. 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 penggunaan Paramiko untuk komunikasi melalui proxy server, diperlukan mekanisme tambahan untuk menangani gangguan koneksi secara lebih efektif.

5.2 Saran

Adapun saran untuk pengembangan selanjutnya adalah sebagai berikut:

1. Menggunakan *webscoket* untuk mengambil dan memperbarui secara *realtime*, lebih disarankan dibandingkan hanya menggunakan API.

2. Menggunakan API untuk mengambil *volume* jumlah data yang besar terbukti dinilai kurang baik karena dapat meningkatkan latensi dan disarankan menambahkan fitur paginasi dan filter untuk mengurangi latensi.
3. Disarankan menambahkan *middlware* selain token autentikasi dan *corsheaders* untuk keamanan API.

TINJAUAN 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 Kinerja Library Paramiko dan Netmiko Dalam Proses Otomasi Jaringan," *Jurnal Nasional Informatika dan Teknologi Jaringan,* vol. 5, no. 1, pp. 1-8, 2020.

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

[4] 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.

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

[6] Mark Masse, REST API Design Rule Book, Cambrige: O'Reilly Media, Inc., 2012.

[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] 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.

[10] 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.

[11] 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.

[12] Eli Nurhayati, Agussalim, "Rancang Bangun Back-end API pada Aplikasi Mobile AyamHub Menggunakan Framework Node JS Express," *Jurnal Sistem dan Teknologi Informasi,* vol. 11, no. 3, pp. 524-531, Juli 2023.

[13] Wira Hadinata, Lilis Stianingsih, "Analisis Perbandingan Performa Restful Api Antara Express.js Dengan Laravel Framework Dengan Jmeter," *Jurnal Informatika dan Teknik Elektro Terapan,* vol. 12, no. 1, pp. 532-540, Januari 2024.

[14] Ida Ayu Kaniya Pradnya Paramitha, Dewa Made Wiharta, I Made Arsa Suyadnya, "Perancangan Dan Implementasi Restful Api Pada Sistem Informasi Manajemen Dosen Universitas Udayana," *Spektrum,* vol. 9, no. 3, pp. 1-9, September 2022.

[15] Romi Choirudin, Ahmat Adil, "Implementasi Rest Api Web Service Dalam Membangun Aplikasi Multiplatform Untuk Usaha Jasa," *Jurnal Matrik,* vol. 18, no. 2, pp. 284-293, Mei 2019.

[16] A. Noertjahyana, "Studi Analisis Rapid Application Development Sebagai Salah Satu Alternatif Metode Pengembangan Perangkat Lunak," *Jurnal Informatika,* vol. 3, no. 2, pp. 74-79, November 2002.

[17] Gede Surya Mahendra , I Kadek Andy Asmarajaya, "Evaluation Using Black Box Testing and System Usability Scale in the Kidung Sekar Madya Application," *Sinkron (Jurnal dan Penelitian Teknik Informatika),* vol. 6, no. 4, pp. 2292-2302, Oktober 2022.

[18] Asep Rizki Maulana, Alam Rahmatulloh, "Websocket untuk Optimasi Kecepatan Data Transfer pada Real Time Chatting," *Innovation in Research of Informatics,* vol. 1, no. 1, pp. 10-11, Maret 2019.

[19] Rheza Adhyatmaka Wiryawan, Nur Rohman Rosyid, "Pengembangan Aplikasi Otomatisasi Administrasi Jaringan Berbasis Website Menggunakan Bahasa Pemrograman Python," *Jurnal SIMETRIS,,* vol. 10, no. 2, pp. 2549-3108 , November 2019.

[20] Hary Sabita, Riko Herwanto, Yuli Syafitri, Bagus Dwi Prasetyo, "Pengembangan Aplikasi Akreditasi Program Studi Berbasis Framework Django," *Jurnal Informatika,* vol. 22, no. 1, pp. 33-37, Juni 2022.

LAMPIRAN