

SKRIPSI

PENGEMBANGAN APLIKASI DESKTOP PEMERIKSAAN
TAUTAN RUSAK PADA SITUS WEB



Ade Rimbo Spencher

NPM: 6182001060

PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INFORMASI DAN SAINS
UNIVERSITAS KATOLIK PARAHYANGAN
2025

UNDERGRADUATE THESIS

**DEVELOPMENT OF A DESKTOP APPLICATION FOR
BROKEN LINK CHECKING IN WEBSITES**



Ade Rimbo Spencher

NPM: 6182001060

**DEPARTMENT OF INFORMATICS
FACULTY OF INFORMATION TECHNOLOGY AND SCIENCES
PARAHYANGAN CATHOLIC UNIVERSITY
2025**

ABSTRAK

«Tuliskan abstrak anda di sini, dalam bahasa Indonesia»

Kata-kata kunci: «Tuliskan di sini kata-kata kunci yang anda gunakan, dalam bahasa Indonesia»

ABSTRACT

«Tuliskan abstrak anda di sini, dalam bahasa Inggris»

Keywords: «Tuliskan di sini kata-kata kunci yang anda gunakan, dalam bahasa Inggris»

DAFTAR ISI

DAFTAR ISI	ix
DAFTAR GAMBAR	xi
1 PENDAHULUAN	1
1.1 Latar Belakang	1
1.2 Rumusan Masalah	3
1.3 Tujuan	3
1.4 Batasan Masalah	3
1.5 Metodologi	3
1.6 Sistematika Pembahasan	4
2 LANDASAN TEORI	5
2.1 HTTP [1]	5
2.1.1 Metode HTTP	5
2.1.2 Kode Status HTTP	6
2.2 URI [2]	10
2.2.1 Struktur Sintaks URL	11
2.2.2 Kategori Karakter dan <i>Percent-Encoding</i>	11
2.2.3 Referensi Absolut dan Relatif	12
2.3 HTML [3]	12
2.3.1 Struktur Dasar HTML	12
2.3.2 Elemen Semantik dan Non-Semantik	13
2.3.3 Atribut Global dan Spesifik	13
2.4 <i>Web Crawling</i> [4]	14
2.4.1 Algoritma <i>Crawling</i>	14
2.4.2 Jenis-Jenis <i>Crawler</i>	15
2.4.3 Tantangan Implementasi	15
2.4.4 Etika dan Kepatuhan	16
2.5 JavaFX [5]	17
2.5.1 Siklus Hidup Aplikasi	17
2.5.2 Stage, Scene, dan Node	18
2.5.3 Komponen FXML	19
2.5.4 Controller, Property, dan Binding	19
2.6 Jsoup	20
2.6.1 Kelas Utama	21
2.6.2 Contoh Kode Program	23
2.7 Java HTTP Client API	24
2.7.1 <code>HttpClient</code>	24
2.7.2 <code>HttpRequest</code>	26
2.7.3 <code>HttpResponse</code>	26
2.7.4 <code>HttpHeaders</code>	27
2.7.5 Contoh Kode Program	28

3 ANALISIS	29
3.1 Analisis Masalah	29
3.1.1 Tautan Rusak	29
3.1.2 <i>Web Crawling</i>	31
3.2 Analisis Sistem Serupa	33
3.2.1 Dead Link Checker	34
3.2.2 Broken Link Checker	36
3.3 Analisis Kebutuhan	39
3.3.1 Kebutuhan Fungsional	41
3.3.2 Kebutuhan Non-Fungsional	41
3.4 Analisis Teknologi yang Digunakan	42
3.4.1 JavaFX	42
3.4.2 Jsoup	43
3.4.3 Java HTTP Client API	44
4 PERANCANGAN	45
4.1 Perancangan Kelas	45
4.2 Perancangan Alur	48
4.3 Perancangan Antarmuka	50
5 IMPLEMENTASI DAN PENGUJIAN	55
5.1 Implementasi	55
5.1.1 Lingkungan Implementasi	55
5.1.2 Implementasi Kelas	55
5.1.3 Implementasi Antarmuka	55
5.2 Pengujian	55
5.2.1 Pengujian Fungsional	55
5.2.2 Pengujian Eksperimental	55
5.2.3 Analisis Hasil Pengujian	55
6 KESIMPULAN DAN SARAN	57
6.1 Kesimpulan	57
6.2 Saran	57
DAFTAR REFERENSI	59
A KODE PROGRAM	61
B HASIL EKSPERIMENT	63

DAFTAR GAMBAR

1.1	Tampilan Broken Link Checker	2
3.1	Antarmuka Dead Link Checker	34
3.2	Antarmuka Broken Link Checker	37
3.3	Diagram <i>use case</i> aplikasi	40
4.1	Diagram kelas aplikasi	49
4.2	Sequence diagram proses pemeriksaan tautan	50
4.3	Rancangan Halaman Utama	52
4.4	Rancangan Jendela Detail Broken Link	53

¹

BAB 1

²

PENDAHULUAN

³ 1.1 Latar Belakang

⁴ Situs web merupakan salah satu sarana utama dalam penyebaran informasi dan representasi identitas
⁵ institusi di era digital. Dalam berbagai sektor, termasuk perguruan tinggi, situs web digunakan
⁶ sebagai kanal resmi untuk menyampaikan informasi secara luas. Perguruan tinggi memanfaatkan
⁷ situs web untuk menyediakan beragam konten seperti kalender akademik, pengumuman penting,
⁸ publikasi ilmiah, dokumentasi kegiatan kampus, serta informasi administratif lainnya. Situs web
⁹ juga menjadi titik awal interaksi antara institusi dengan calon mahasiswa, mitra kerja, media, dan
¹⁰ masyarakat umum.

¹¹ Pentingnya situs web pada perguruan tinggi juga tercermin dari adanya berbagai sistem pemeringkatan non-akademis yang menjadikan situs web resmi perguruan tinggi sebagai objek penilaian.
¹² Dua contoh sistem pemeringkatan tersebut adalah Webometrics dan uniRank. Webometrics¹
¹³ merupakan sistem pemeringkatan yang dikembangkan oleh Cybermetrics Lab, sebuah kelompok
¹⁴ riset di bawah Consejo Superior de Investigaciones Científicas (CSIC) Spanyol. Sistem ini menilai
¹⁵ kehadiran dan visibilitas situs web akademik di internet sebagai cerminan dari kinerja dan dampak
¹⁶ institusi dalam ranah digital. Namun, sejak April 2025 laman resmi Webometrics sudah tidak dapat
¹⁷ diakses. Sementara itu, uniRank² adalah sistem pemeringkatan yang berfokus pada popularitas
¹⁸ dan kehadiran daring situs web perguruan tinggi. Penilaian pada uniRank dilakukan menggunakan
¹⁹ empat metrik independen, yaitu *Majestic Referring Domains* (55%), *Similarweb Global Rank* (35%),
²⁰ *Moz Domain Authority* (5%), dan *Majestic Trust Flow* (5%). *Majestic Referring Domains* mengukur
²¹ jumlah domain eksternal unik yang memberikan tautan menuju situs web resmi perguruan tinggi,
²² dengan mempertimbangkan kualitas tautan melalui ambang batas *Trust Flow* tertentu. Sementara
²³ itu, *Similarweb Global Rank* mencerminkan tingkat popularitas situs berdasarkan estimasi trafik
²⁴ global yang diterima dari berbagai sumber pengunjung. Berdasarkan kedua sistem tersebut, dapat
²⁵ disimpulkan bahwa visibilitas dan pengalaman pengguna merupakan dua aspek yang penting dari
²⁶ situs web perguruan tinggi dan perlu diperhatikan secara berkelanjutan.

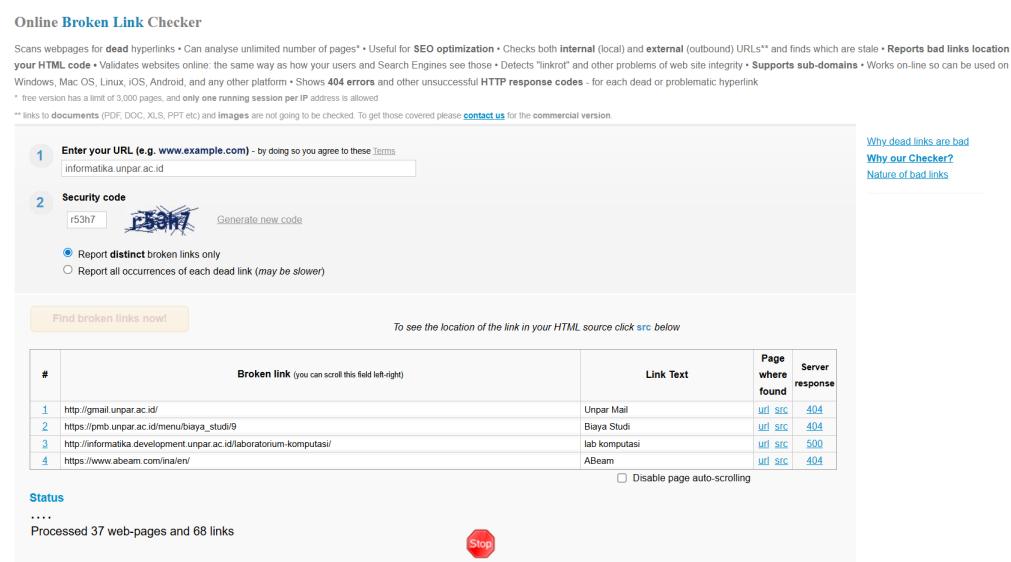
²⁷ Salah satu tantangan dalam menjaga visibilitas dan pengalaman pengguna dalam mengakses situs
²⁸ web perguruan tinggi adalah keberadaan tautan rusak. Tautan rusak terjadi ketika suatu tautan
²⁹ mengarah ke sumber daya yang sudah tidak tersedia atau tidak dapat diakses, sehingga menghasilkan
³⁰ pesan kesalahan seperti 404 Not Found atau 500 Internal Server Error. Keberadaan tautan
³¹ rusak dapat menghambat alur navigasi, menurunkan kepercayaan pengguna terhadap isi situs, serta

¹<https://webometrics.info> (Diakses pada 7 April 2025)

²<https://www.unirank.org/about> (Diakses pada 20 Juli 2025)

berdampak negatif terhadap peringkat situs dalam mesin pencari dan sistem pemeringkatan daring, seperti uniRank dan Webometrics. Oleh karena itu, pemantauan dan pemeliharaan tautan secara berkala merupakan langkah penting dalam pengelolaan situs web institusi pendidikan tinggi.

Secara umum, terdapat dua pendekatan dalam memeriksa keberadaan tautan rusak pada situs web, yaitu secara manual dan dengan bantuan perangkat lunak. Pemeriksaan manual menjadi tidak praktis ketika situs memiliki ratusan bahkan ribuan halaman. Untuk mengatasi hal ini, tersedia berbagai layanan daring seperti Broken Link Checker³. Broken Link Checker merupakan alat berbasis web yang secara otomatis memindai halaman-halaman dalam sebuah situs web dan mendeteksi tautan rusak di dalamnya. Gambar 1.1 menunjukkan hasil pemindaian pada situs web Informatika UNPAR menggunakan layanan Broken Link Checker, di mana terdeteksi beberapa tautan rusak yang tersebar di berbagai halaman. Dalam laporannya, Broken Link Checker menampilkan informasi penting seperti URL dari tautan rusak yang ditemukan, teks yang menjadi jangkar (*anchor text*) dari tautan tersebut, halaman tempat tautan itu ditemukan, serta respons dari server dalam bentuk kode status HTTP seperti 404 atau 500. Layanan ini juga menyediakan tautan langsung menuju bagian kode sumber tautan rusak ditemukan untuk mempermudah pelacakan lokasi tautan rusak dalam dokumen HTML.



Gambar 1.1: Tampilan Broken Link Checker

Dalam penelitian ini akan dikembangkan sebuah perangkat lunak berbasis desktop yang dapat digunakan untuk memeriksa keberadaan tautan rusak pada situs web secara otomatis. Perangkat lunak ini akan menerima masukan berupa URL alamat situs web, kemudian menelusuri halaman-halaman yang saling terhubung di dalamnya untuk mengidentifikasi seluruh tautan yang terdapat pada situs tersebut. Selanjutnya, setiap tautan akan diperiksa dan dievaluasi respons yang diberikan oleh server. Berdasarkan hasil pemeriksaan tersebut, perangkat lunak akan menghasilkan laporan yang memuat informasi mengenai tautan-tautan yang terdeteksi rusak. Untuk mempermudah pengguna dalam menjalankan proses pemeriksaan dan membaca hasilnya, perangkat lunak ini akan dilengkapi dengan antarmuka pengguna grafis yang intuitif.

³<https://www.brokenlinkcheck.com/> (Diakses pada 20 Juli 2025)

1 1.2 Rumusan Masalah

- 2 Berdasarkan latar belakang yang telah diuraikan, permasalahan yang akan dibahas dalam penelitian
3 ini dirumuskan sebagai berikut:
- 4 1. Bagaimana mengembangkan perangkat lunak berbasis desktop yang dapat melakukan pemeriksaan
5 tautan rusak pada situs web?
- 6 2. Bagaimana melakukan pengujian perangkat lunak pemeriksa tautan rusak dan membandingkan
7 hasil yang diperoleh dengan perangkat lunak serupa?

8 1.3 Tujuan

- 9 Tujuan dari penelitian ini adalah sebagai berikut:
- 10 1. Mengembangkan perangkat lunak berbasis desktop yang mampu melakukan pemeriksaan
11 terhadap tautan rusak pada situs web.
- 12 2. Melakukan pengujian terhadap perangkat lunak pemeriksa tautan rusak yang dikembangkan,
13 serta membandingkan hasilnya dengan perangkat lunak serupa.

14 1.4 Batasan Masalah

- 15 Batasan masalah pada tugas akhir ini adalah sebagai berikut:
- 16 1. Proses *web crawling* hanya dilakukan pada halaman-halaman situs yang berada dalam satu
17 *host* yang sama.
- 18 2. Pemeriksaan tautan hanya mencakup tautan yang secara eksplisit tercantum dalam dokumen
19 HTML yang diterima langsung dari *server* pada saat halaman dimuat. Tautan yang
20 dimunculkan secara dinamis melalui eksekusi JavaScript pada sisi klien tidak termasuk dalam
21 cakupan pemeriksaan.

22 1.5 Metodologi

- 23 Metodologi penelitian yang digunakan pada tugas akhir ini adalah sebagai berikut:
- 24 1. Melakukan studi literatur terhadap protokol HTTP, URI, situs web dan *web crawling*.
- 25 2. Melakukan studi literatur dan eksperimen terhadap Jsoup, Java HttpClient, JavaFX dan
Gradle.
- 26 3. Menganalisis permasalahan pemeriksaan tautan rusak pada situs web, meninjau perangkat
lunak serupa, merumuskan kebutuhan fungsional dan non-fungsional, serta menganalisis
teknologi yang akan digunakan dalam pengembangan aplikasi.
- 27 4. Merancang kelas-kelas utama dan antarmuka pengguna perangkat lunak.
- 28 5. Mengimplementasikan perangkat lunak pemeriksa tautan rusak situs web.
- 29 6. Melakukan pengujian terhadap beberapa situs web dan membandingkan hasilnya dengan
perangkat lunak sejenis untuk mengevaluasi hasil dan kemampuannya dalam pemeriksaan
tautan rusak.
- 30 7. Menulis dokumen tugas akhir.

1.6 Sistematika Pembahasan

2 Sistematika pembahasan dalam penelitian ini disusun sebagai berikut:

3 1. Bab 1 Pendahuluan

4 Bab ini berisi latar belakang dilakukannya penelitian, rumusan masalah, tujuan penelitian,
5 batasan masalah, metodologi yang digunakan, serta sistematika pembahasan.

6 2. Bab 2 Landasan Teori

7 Bab ini membahas teori-teori yang menjadi dasar dalam pengembangan perangkat lunak
8 pemeriksa tautan rusak pada situs web. Pembahasan mencakup protokol HTTP, URI, konsep
9 dasar situs web, *web crawling*, serta pustaka dan teknologi yang digunakan seperti Jsoup,
10 Java HttpClient, JavaFX dan Gradle.

11 3. Bab 3 Analisis

12 Bab ini membahas analisis permasalahan pemeriksaan tautan rusak pada situs web, peninjauan
13 perangkat lunak serupa, perumusan kebutuhan fungsional dan non-fungsional, serta analisis
14 teknologi yang digunakan dalam pengembangan aplikasi.

15 4. Bab 4 Perancangan

16 Bab ini membahas perancangan perangkat lunak berdasarkan hasil analisis yang sudah dibuat.
17 Perancangan meliputi perancangan kelas dan perancangan antarmuka pengguna.

18 5. Bab 5 Implementasi dan Pengujian

19 Bab ini memaparkan hasil implementasi perangkat lunak pemeriksa tautan rusak berdasarkan
20 perancangan yang sudah dibuat. Selain itu, bab ini juga memaparkan hasil pengujian
21 fungsional dan eksperimental serta analisis terhadap hasil pengujian.

22 6. Bab 6 Kesimpulan dan Saran

23 Bab ini berisi kesimpulan dari hasil pengembangan perangkat lunak pemeriksa tautan rusak
24 pada situs web, serta saran pengembangan lebih lanjut yang didasarkan pada hasil analisis
25 pengujian yang sudah dilakukan.

1

BAB 2

2

LANDASAN TEORI

- 3 Bab ini membahas teori dan teknologi yang mendasari pengembangan perangkat lunak pemeriksa
4 tautan rusak, mencakup protokol HTTP, URI, konsep dasar situs web, *web crawling*, serta pustaka
5 dan teknologi yang digunakan seperti Jsoup, Java HTTP Client API, dan JavaFX.

6 2.1 HTTP [1]

7 *Hypertext Transfer Protocol* (HTTP) adalah protokol tingkat aplikasi yang menjadi dasar komunikasi
8 pada arsitektur *World Wide Web*. HTTP bekerja dengan pola komunikasi berbasis pesan, di mana
9 sebuah *client* mengirimkan *request* dan *server* menanggapinya dengan *response*. Protokol ini bersifat
10 *stateless*, artinya setiap *request* dapat dipahami secara terpisah tanpa ketergantungan pada interaksi
11 sebelumnya, sehingga *server* tidak diwajibkan menyimpan konteks antar *request*.

12 Tujuan dari sebuah *request* adalah *resource*, yang umumnya diidentifikasi melalui URI (lihat
13 Subbab 2.2). HTTP tidak membatasi apa yang dimaksud dengan *resource*, melainkan hanya
14 menyediakan antarmuka untuk berinteraksi dengannya. Informasi mengenai *resource* tersebut
15 disampaikan dalam bentuk *representation*, yaitu data beserta metadata yang mencerminkan keadaan
16 *resource* pada waktu tertentu dan dapat ditransmisikan melalui protokol.

17 Untuk meningkatkan kinerja, HTTP juga memanfaatkan *caches*. Sebuah *cache* menyimpan
18 *response* sebelumnya yang bersifat *cacheable*, sehingga *request* yang sama di kemudian hari dapat
19 dijawab lebih cepat tanpa perlu menghubungi kembali *origin server*. *Origin server* adalah program
20 yang dapat menghasilkan *response* otoritatif untuk sebuah *resource*. Mekanisme ini membantu
21 mengurangi waktu *response* sekaligus menekan konsumsi *bandwidth*.

22 Secara umum, HTTP dijalankan di atas protokol TCP dengan port standar 80. Untuk komunikasi
23 yang membutuhkan perlindungan, digunakan skema **https**, yaitu HTTP yang berjalan di atas
24 *Transport Layer Security* (TLS) pada port standar 443. Melalui HTTPS, komunikasi memperoleh
25 jaminan kerahasiaan, autentikasi, dan integritas data.

26 2.1.1 Metode HTTP

27 Metode HTTP berfungsi untuk menunjukkan tujuan dari *request* yang dibuat oleh *client* dan
28 hasil sukses apa yang diharapkan dari *request* tersebut. Metode HTTP memiliki beberapa sifat
29 umum, diantaranya adalah *safe* dan *idempotent*. Sebuah metode dikatakan *safe* apabila semantik
30 yang didefinisikan bersifat *read-only*, yaitu *client* tidak meminta dan tidak mengharapkan adanya
31 perubahan keadaan pada *origin server* akibat penerapan metode tersebut. Metode yang termasuk

1 *safe* adalah GET, HEAD, OPTIONS, dan TRACE. Sebuah metode disebut *idempotent* apabila dampak
2 yang dimaksudkan pada *server* dari beberapa *request* identik, sama dengan dampak dari satu
3 *request*. Metode PUT, DELETE, serta seluruh metode aman adalah *idempotent*. Berikut ini adalah
4 daftar beberapa metode HTTP:

- 5 • **GET**: Metode ini digunakan untuk meminta transfer representasi terkini dari sumber daya
target.
- 7 • **HEAD**: Metode ini identik dengan GET, tetapi *server* tidak boleh mengirimkan konten dalam
response. HEAD digunakan untuk memperoleh metadata dari representasi yang dipilih tanpa
9 harus mentransfer data representasi itu sendiri.
- 10 • **POST**: Metode ini digunakan untuk meminta sumber daya target memproses representasi
11 yang disertakan dalam *request* sesuai dengan semantik khusus yang dimiliki oleh sumber daya
12 tersebut.
- 13 • **PUT**: Metode ini digunakan untuk meminta agar keadaan dari sumber daya target dibuat
14 atau diganti dengan keadaan yang ditentukan oleh representasi yang disertakan dalam isi
15 pesan *request*.
- 16 • **DELETE**: Metode ini digunakan untuk meminta agar *origin server* menghapus asosiasi
17 antara sumber daya target dengan fungsionalitasnya saat ini.
- 18 • **OPTIONS**: Metode ini digunakan untuk meminta informasi mengenai opsi komunikasi yang
19 tersedia bagi sumber daya target, baik pada *origin server* maupun perantara. Metode ini
20 memungkinkan *client* mengetahui opsi dan/atau persyaratan yang terkait dengan sebuah
21 sumber daya, atau kemampuan dari sebuah *server*, tanpa menyiratkan adanya tindakan
22 terhadap sumber daya tersebut.

23 2.1.2 Kode Status HTTP

24 Kode status HTTP adalah bagian dari baris awal pada *response server* yang menunjukkan hasil
25 pemrosesan terhadap suatu *request*. Kode ini terdiri dari tiga digit numerik dan dikelompokkan
26 ke dalam lima kelas utama berdasarkan digit pertamanya: informasi (1xx), keberhasilan (2xx),
27 pengalihan (3xx), kesalahan dari *client* (4xx), dan kesalahan dari *server* (5xx).

28 *Informational 1xx*

29 Kode-kode pada kelas ini menunjukkan bahwa *request* telah diterima dan sedang diproses, tetapi
30 belum ada *response* final.

- 31 • **100 (Continue)**: Menunjukkan bahwa bagian awal dari *request* telah diterima dan belum
32 ditolak oleh *server*. *Server* bermaksud untuk mengirimkan *response* akhir setelah seluruh
33 *request* diterima dan diproses.
- 34 • **101 (Switching Protocols)**: Menunjukkan bahwa *server* memahami dan bersedia memenuhi
35 *request* *client* untuk beralih protokol aplikasi pada koneksi yang sama.
- 36 • **102 (Processing)**: Menunjukkan bahwa *server* telah menerima *request* sepenuhnya, namun
37 pemrosesan terhadap *request* tersebut belum selesai. [6]
- 38 • **103 (Early Hints)**: Digunakan oleh *server* untuk memberikan *response* awal melalui *header*
39 kepada *client* sebelum *response* akhir tersedia. [7]

1 ***Successful 2xx***

2 Kode-kode pada kelas ini menunjukkan bahwa *request* telah diterima, dipahami, dan diproses dengan
3 sukses.

- 4 • **200 (OK)**: Menunjukkan bahwa *request* berhasil diproses.
- 5 • **201 (Created)**: Menunjukkan bahwa *request* berhasil dan menghasilkan satu atau lebih
6 sumber daya baru.
- 7 • **202 (Accepted)**: Menunjukkan bahwa *request* telah diterima untuk diproses, tetapi pemro-
8 sesan belum selesai.
- 9 • **203 (Non-Authoritative Information)**: Menunjukkan bahwa *request* berhasil, tetapi
10 konten yang dikirim telah dimodifikasi dari *response* 200 (*OK*) oleh *transforming proxy*.
11 *Transforming proxy* adalah *proxy* yang melakukan modifikasi terhadap representasi pesan, baik
12 dengan cara mengubah format, menambahkan informasi, maupun menghapus sebagian konten,
13 sehingga representasi yang dikirimkan kepada *client* tidak identik dengan yang dikirimkan
14 oleh *origin server*.
- 15 • **204 (No Content)**: Menunjukkan bahwa *request* berhasil, tetapi tidak ada konten tambahan
16 untuk dikirim dalam *response*.
- 17 • **205 (Reset Content)**: Menunjukkan bahwa *server* telah berhasil memenuhi *request* dan
18 menginginkan agar *user agent* mereset tampilan dokumen yang menyebabkan *request* dikirim,
19 ke keadaan awal sebagaimana diterima dari *origin server*.
- 20 • **206 (Partial Content)**: Menunjukkan bahwa *server* berhasil memenuhi *range request* dengan
21 mengirimkan sebagian dari representasi sumber daya. Klien harus memeriksa **Content-Type**
22 dan **Content-Range** untuk mengetahui bagian mana yang dikirim dan apakah diperlukan
23 *request* tambahan. *Range request* didefinisikan sebagai *request* HTTP yang menggunakan
24 **header Range** untuk meminta sebagian dari representasi data.
- 25 • **207 (Multi-Status)**: Digunakan untuk memberikan status terhadap beberapa operasi
26 independen dalam satu *request*. [8]
- 27 • **226 (IM Used)**: Menunjukkan bahwa *server* telah berhasil memenuhi *request* GET terhadap
28 suatu sumber daya, dan representasi yang dikembalikan merupakan hasil dari satu atau lebih
29 *instance manipulations* yang diterapkan pada *instance* saat ini. *Instance manipulation* adalah
30 operasi terhadap satu atau lebih *instance* yang dapat menghasilkan representasi *instance*
31 dikirimkan dari *server* ke klien dalam bentuk bagian-bagian terpisah atau melalui lebih dari
32 satu pesan *response*. [9]

33 ***Redirection 3xx***

34 Kode-kode pada kelas ini menunjukkan bahwa *client* harus melakukan langkah tambahan untuk
35 menyelesaikan *request*, seperti mengikuti *redirect*.

- 36 • **300 (Multiple Choices)**: Menunjukkan bahwa sumber daya target memiliki lebih dari
37 satu representasi, masing-masing dengan *identifier* yang lebih spesifik. Informasi mengenai
38 alternatif tersebut diberikan agar pengguna atau *user agent* dapat memilih representasi yang
39 diinginkan dengan mengarahkan permintaannya ke salah satu dari *identifier* tersebut.
- 40 • **301 (Moved Permanently)**: Menunjukkan bahwa sumber daya target telah diberikan URI
41 baru yang bersifat permanen, dan semua referensi di masa depan sebaiknya menggunakan

1 URI baru tersebut.

- 2 • **302 (*Found*)**: Menunjukkan bahwa sumber daya target sementara berada di bawah URI
3 yang berbeda. Karena lokasi tersebut dapat berubah, *client* tetap sebaiknya menggunakan
4 URI asli untuk *request* di masa depan.
- 5 • **303 (*See Other*)**: Menunjukkan bahwa *server* mengarahkan *user agent* ke sumber daya
6 lain, sebagaimana ditentukan dalam *header Location*, yang dimaksudkan untuk memberikan
7 *response* tidak langsung terhadap *request* asli.
- 8 • **304 (*Not Modified*)**: Menunjukkan bahwa *server* telah menerima *conditional request* untuk
9 metode GET atau HEAD yang seharusnya menghasilkan *response* 200 (*OK*), tetapi kondisi yang
10 diberikan bernilai salah. Artinya, *client* sudah memiliki representasi yang valid, sehingga
11 *server* tidak perlu mengirim ulang. Klien dapat menggunakan salinan yang sudah dimiliki
12 seolah-olah *server* memberikan *response* 200 (*OK*).
- 13 • **307 (*Temporary Redirect*)**: Menunjukkan bahwa sumber daya target sementara berada di
14 bawah URI yang berbeda.
- 15 • **308 (*Permanent Redirect*)**: Menunjukkan bahwa sumber daya target telah dipindahkan
16 secara permanen ke URI baru, dan semua referensi berikutnya sebaiknya menggunakan URI
17 baru tersebut.

18

Client Error 4xx

19 Kode-kode pada kelas ini menunjukkan bahwa telah terjadi kesalahan di sisi *client*. Berikut adalah
20 daftar kode pada kelas ini:

- 21 • **400 (*Bad Request*)**: Menunjukkan bahwa *server* tidak dapat atau tidak mau memproses
22 *request* karena dianggap sebagai kesalahan dari *client*, seperti sintaks *request* salah, format
23 pesan tidak valid, atau rute *request* yang menyesatkan.
- 24 • **401 (*Unauthorized*)**: Menunjukkan bahwa *request* tidak dapat dijalankan karena tidak
25 memiliki kredensial autentikasi yang sah untuk sumber daya yang diminta.
- 26 • **402 (*Payment Required*)**: Kode ini disediakan untuk penggunaan di masa depan.
- 27 • **403 (*Forbidden*)**: Menunjukkan bahwa *server* memahami *request* tetapi menolak untuk
28 memenuhinya.
- 29 • **404 (*Not Found*)**: Menunjukkan bahwa sumber daya yang diminta tidak ditemukan.
- 30 • **405 (*Method Not Allowed*)**: Menunjukkan bahwa metode HTTP pada *request* dikenali
31 oleh *server*, tetapi tidak diizinkan untuk digunakan pada sumber daya tersebut.
- 32 • **406 (*Not Acceptable*)**: Menunjukkan bahwa sumber daya target tidak memiliki representasi
33 yang sesuai dengan preferensi *client* berdasarkan *header negosiasi konten*.
- 34 • **407 (*Proxy Authentication Required*)**: Mirip dengan kode status 401, tetapi digunakan
35 ketika *client* harus melakukan autentikasi terlebih dahulu kepada *proxy*.
- 36 • **408 (*Request Timeout*)**: Menunjukkan bahwa *server* tidak menerima pesan *request* yang
37 lengkap dalam jangka waktu yang sudah disiapkannya untuk menunggu.
- 38 • **409 (*Conflict*)**: Menunjukkan bahwa *request* tidak dapat diselesaikan karena terjadi konflik
39 dengan keadaan sumber daya target saat ini.
- 40 • **410 (*Gone*)**: Menunjukkan bahwa akses ke sumber daya target sudah tidak tersedia lagi di
41 *origin server*, dan kondisi ini kemungkinan bersifat permanen.

- **411 (Length Required)**: Menunjukkan bahwa *server* menolak menerima *request* yang tidak memiliki *header Content-Length*.
- **412 (Precondition Failed)**: Menunjukkan bahwa satu atau lebih kondisi yang dikirim dalam *header request* bernilai salah ketika diperiksa oleh *server*.
- **413 (Content Too Large)**: Menunjukkan bahwa *server* menolak memproses *request* karena ukuran konten lebih besar daripada yang bersedia atau mampu diproses *server*.
- **414 (URI Too Long)**: Menunjukkan bahwa *server* menolak memproses *request* karena URI target terlalu panjang untuk ditafsirkan.
- **415 (Unsupported Media Type)**: Menunjukkan bahwa *server* menolak memproses *request* karena isi *request* berada dalam format yang tidak didukung oleh metode pada sumber daya target.
- **416 (Range Not Satisfiable)**:
- **417 (Expectation Failed)**: Menunjukkan bahwa ekspektasi yang ditentukan dalam *header Expect* pada *request*, tidak dapat dipenuhi oleh setidaknya salah satu *server* yang menerima *request*.
- **421 (Misdirected Request)**: Menunjukkan bahwa *request* diarahkan ke *server* yang tidak mampu atau tidak mau memberikan *response* otoritatif untuk URI target.
- **422 (Unprocessable Content)**: Menunjukkan bahwa *server* memahami jenis konten *request* dan sintaks *request* benar, tetapi *server* tidak dapat memproses instruksi di dalamnya.
- **423 (Locked)**: Menunjukkan bahwa sumber atau tujuan dari suatu metode sedang terkunci. [8]
- **424 (Failed Dependency)**: Menunjukkan bahwa metode tidak dapat dijalankan pada sumber daya karena tindakan yang menjadi prasyarat telah gagal. [8]
- **425 (Too Early)**: Menunjukkan bahwa *server* menolak memproses *request* karena khawatir *request* tersebut dapat diulang (*replayed*). [10]
- **426 (Upgrade Required)**: Menunjukkan bahwa *server* menolak menjalankan *request* menggunakan protokol saat ini, tetapi bersedia melakukannya setelah *client* beralih ke protokol lain.
- **428 (Precondition Required)**: Menunjukkan bahwa *server* mewajibkan *request* disertai kondisi tertentu. [11]
- **429 (Too Many Requests)**: Menunjukkan bahwa *client* telah mengirim terlalu banyak *request* dalam jangka waktu tertentu. [11]
- **431 (Request Header Fields Too Large)**: Menunjukkan bahwa *server* menolak *request* karena ukuran *header* terlalu besar. [11]
- **451 (Unavailable For Legal Reasons)**: Sumber daya dibatasi karena alasan hukum. [12]

35 **Server Error 5xx**

- 36 Kode-kode pada kelas ini menunjukkan bahwa *server* menyadari bahwa terjadi kesalahan di sisi
37 *server*. Berikut adalah daftar kode pada kelas ini:
- **500 (Internal Server Error)**: Menunjukkan bahwa *server* mengalami kondisi tak terduga yang menghalangi *server* untuk dapat memenuhi.
 - **501 (Not Implemented)**: Menunjukkan bahwa *server* tidak mendukung fungsionalitas yang diperlukan untuk memenuhi *request* dari *client*. Kode ini digunakan ketika *server* tidak

1 mengenali metode HTTP yang digunakan, dan tidak memiliki kemampuan untuk mendukung
2 metode tersebut pada sumber daya mana pun.

- 3 • **502 (Bad Gateway)**: Menunjukkan bahwa *server* ketika bertindak sebagai *gateway* atau
4 *proxy*, menerima *response* tidak valid dari *server* lain yang diaksesnya saat mencoba
5 memenuhi *request*.
- 6 • **503 (Service Unavailable)**: Menunjukkan bahwa *server* untuk sementara tidak dapat
7 menangani *request*. Kode ini digunakan ketika *server* sedang kelebihan beban atau sedang
8 dilakukan pemeliharaan. *Server* dapat menyertakan *header Retry-After* untuk memberi
9 tahu *client* berapa lama sebaiknya menunggu sebelum mencoba lagi *request*.
- 10 • **504 (Gateway Timeout)**: Menunjukkan bahwa *server* ketika bertindak sebagai *gateway*
11 atau *proxy*, tidak menerima *response* tepat waktu dari *server* lain yang perlu diakses untuk
12 menyelesaikan *request*.
- 13 • **505 (HTTP Version Not Supported)**: Menunjukkan bahwa *server* tidak mendukung atau
14 menolak mendukung versi HTTP yang digunakan dalam *request*.
- 15 • **506 (Variant Also Negotiates)**: Menunjukkan bahwa *server* memiliki kesalahan konfigu-
16 rasi internal. Hal ini terjadi ketika sumber daya yang dipilih sebagai varian ternyata juga
17 dikonfigurasi untuk melakukan *transparent content negotiation*, yaitu mekanisme untuk secara
18 otomatis memilih versi terbaik dari sebuah sumber daya yang memiliki banyak varian di
19 bawah satu URL. Akibatnya, sumber daya tersebut tidak bisa menjadi titik akhir yang valid
20 dalam proses negosiasi konten. [13]
- 21 • **507 (Insufficient Storage)**: Menunjukkan bahwa metode tidak dapat dijalankan pada
22 sumber daya karena *server* tidak mampu menyimpan representasi yang dibutuhkan untuk
23 menyelesaikan *request* dengan sukses. [8]
- 24 • **508 (Loop Detected)**: Menunjukkan bahwa *server* menghentikan suatu operasi karena
25 mendeteksi adanya *infinite loop* ketika memproses *request* dengan *header Depth:infinity*. [14]
- 26 • **511 (Network Authentication Required)**: Menunjukkan bahwa *client* harus melakukan
27 autentikasi terlebih dahulu untuk mendapatkan akses jaringan. [11]

28 2.2 URI [2]

29 *Uniform Resource Identifier* (URI) adalah string karakter yang digunakan untuk mengidentifikasi
30 suatu sumber daya. Identifikasi tersebut dapat berupa lokasi, nama, atau kombinasi keduanya.
31 Spesifikasi URI ditetapkan dalam RFC 3986 yang mendefinisikan aturan sintaks, komponen, serta
32 mekanisme representasi sumber daya.

33 RFC 3986 mendeskripsikan URI sebagai konsep umum yang memiliki beberapa bentuk. Dua
34 bentuk yang paling dikenal adalah *Uniform Resource Locator* (URL) dan *Uniform Resource Name*
35 (URN). URL adalah URI yang menyertakan informasi lokasi dan mekanisme akses terhadap sumber
36 daya, contohnya <http://unpar.ac.id/index.html>. URN adalah URI yang berfungsi sebagai nama
37 tetap suatu sumber daya tanpa menyertakan lokasi aksesnya, contohnya [urn:isbn:0451450523](#).
38 RFC juga membedakan URI yang bersifat hierarkis dan URI yang bersifat opak. URI hierarkis
39 memiliki komponen seperti *authority* dan *path*, misalnya skema <http>. URI opak tidak dapat
40 diuraikan ke dalam komponen hierarkis, misalnya skema <mailto>.

1 2.2.1 Struktur Sintaks URL

2 URL, sebagai bentuk URI yang bersifat hierarkis, memiliki struktur sintaks yang terdiri atas
3 beberapa komponen. Format umum penulisan URL ditunjukkan sebagai berikut:

4 `scheme : // [user:password@] host [:port]] path [?query] [#fragment]`

5 Komponen-komponen dalam URL dijelaskan sebagai berikut:

6 • **Scheme**

7 Bagian ini dituliskan pada awal URL dan diakhiri dengan tanda titik dua (:). Scheme
8 menunjukkan protokol atau mekanisme akses yang digunakan, contohnya `http`, `https`, `ftp`,
9 atau `mailto`.

10 • **Authority**

11 Bagian ini bersifat opsional dan diawali dengan dua garis miring (//). Authority dapat terdiri
12 atas tiga subkomponen, yaitu `userinfo`, `host`, dan `port`. `Userinfo` berisi informasi pengguna
13 dengan format `user:password@` yang digunakan dalam beberapa skema seperti FTP. `Host`
14 adalah lokasi sumber daya yang dapat berupa nama domain atau alamat IP. `Port` adalah
15 nomor port koneksi, misalnya 80 untuk HTTP atau 443 untuk HTTPS. Jika port tidak
16 dituliskan, maka `port default` dari skema tersebut digunakan.

17 • **Path**

18 Bagian ini menunjukkan jalur menuju sumber daya pada host. Path terdiri atas segmen-
19 segmen yang dipisahkan tanda garis miring (/). Di dalamnya dapat terdapat `dot-segments`,
20 yaitu segmen khusus berupa . yang merujuk ke direktori saat ini dan .. yang merujuk ke
21 direktori induk.

22 • **Query**

23 Bagian ini bersifat opsional dan diawali dengan tanda tanya (?). Query berisi parameter dalam
24 format pasangan `key=value`. Jika terdapat lebih dari satu parameter, maka masing-masing
25 dipisahkan dengan tanda &.

26 • **Fragment**

27 Bagian ini bersifat opsional dan diawali dengan tanda pagar (#). Fragment digunakan untuk
28 merujuk ke bagian tertentu dari sumber daya. Informasi ini tidak dikirim ke server, melainkan
29 diproses oleh agen pengguna seperti browser.

30 2.2.2 Kategori Karakter dan *Percent-Encoding*

31 RFC 3986 menetapkan kategori karakter yang dapat digunakan dalam URI, beserta aturan pengko-
32 deannya.

33 • **Unreserved characters**

34 Karakter ini dapat digunakan langsung tanpa pengkodean tambahan. Termasuk di dalamnya
35 huruf alfabet A sampai Z dan a sampai z, digit angka 0 sampai 9, serta simbol -, _, ., dan ~.

36 • **Reserved characters**

37 Karakter ini memiliki fungsi khusus tergantung pada konteks penggunaannya. RFC membagi
38 reserved characters menjadi dua kelompok. General delimiters meliputi :, /, ?, #, [,], dan
39 @. Subcomponent delimiters meliputi !, \$, &, ', (,), *, +, ,, ;, dan =. Jika karakter ini
40 digunakan di luar konteks yang diperbolehkan, maka harus dikodekan.

1 • **Karakter lain**

2 Karakter non-ASCII, spasi, dan simbol lain yang tidak termasuk dalam kategori di atas tidak
3 dapat digunakan secara langsung. Karakter ini harus dikodekan sebelum dapat dimasukkan
4 ke dalam URI.

5 Pengkodean karakter dilakukan dengan mekanisme *percent-encoding*. Mekanisme ini menggantikan karakter dengan representasi heksadesimalnya dalam format %HH, di mana HH adalah nilai
6 ASCII karakter tersebut. Sebagai contoh, spasi ditulis sebagai %20, tanda kutip ganda sebagai
7 %22, dan tanda pagar sebagai %23. RFC juga mengatur bahwa karakter unreserved yang tidak
8 perlu dikodekan sebaiknya dituliskan dalam bentuk aslinya. Selain itu, digit heksadesimal dalam
9 *percent-encoding* harus ditulis dengan huruf besar.
10

11 **2.2.3 Referensi Absolut dan Relatif**

12 URL dapat berupa referensi absolut maupun relatif. URL absolut mencantumkan seluruh komponen
13 utama termasuk scheme dan authority, sehingga dapat diinterpretasikan secara mandiri. Contohnya
14 adalah <https://unpar.ac.id/page.html>. URL relatif tidak mencantumkan scheme atau authority
15 dan hanya menyertakan path, query, atau fragment. Contohnya adalah `images/logo.png`. Selain
16 itu terdapat pula same-document reference, yaitu URL yang hanya berisi fragment untuk merujuk
17 ke bagian tertentu dalam dokumen yang sama, contohnya `#section2`.

18 RFC 3986 mendefinisikan mekanisme resolusi yang digunakan untuk menyusun URL absolut
19 dari sebuah referensi relatif. Resolusi dilakukan dengan menentukan base URI, menggabungkannya
20 dengan URL relatif, lalu menyederhanakan hasilnya. Salah satu tahap penting dalam resolusi adalah
21 penghapusan dot-segments. Prosedur ini menghilangkan segmen . dan .. agar path hasil resolusi
22 berada dalam bentuk yang ringkas dan tidak ambigu.

23 **2.3 HTML [3]**

24 Hypertext Markup Language (HTML) merupakan bahasa markup standar yang digunakan untuk
25 menyusun dan menampilkan halaman web pada *browser*. HTML didefinisikan sebagai himpunan
26 elemen yang dituliskan dalam bentuk tag, di mana setiap elemen dapat memiliki atribut untuk
27 memberikan informasi tambahan. Dokumen HTML tersusun secara hierarkis dan secara konseptual
28 dipandang sebagai sebuah pohon struktur yang dikenal dengan *Document Object Model* (DOM).

29 Sejak diperkenalkan pada awal 1990-an, HTML telah berkembang melalui beberapa versi. HTML
30 4.01 yang dirilis pada tahun 1999 menjadi salah satu versi yang banyak digunakan dan bertahan
31 cukup lama. Setelah itu muncul XHTML sebagai reformulasi HTML dalam sintaks XML, namun
32 penerapannya terbatas. Versi terbaru adalah HTML5 yang dikembangkan oleh Web Hypertext
33 Application Technology Working Group (WHATWG) dan kemudian diadopsi oleh World Wide Web
34 Consortium (W3C), dengan dukungan yang lebih baik terhadap elemen semantik, multimedia, serta
35 penulisan sintaks yang lebih sederhana.

36 **2.3.1 Struktur Dasar HTML**

37 Sebuah dokumen HTML terdiri dari beberapa bagian utama. Dokumen diawali dengan deklarasi
38 `<!DOCTYPE html>` yang memberi tahu *browser* mengenai standar yang digunakan. Seluruh isi

- 1 dokumen dibungkus dalam elemen `<html>` yang menjadi akar dari semua elemen lain.
- 2 Di dalam `<html>` terdapat dua bagian penting, yaitu `<head>` dan `<body>`. Bagian `<head>` memuat informasi tentang dokumen seperti judul, metadata, dan pemanggilan sumber daya eksternal.
- 3 Bagian `<body>` berisi konten utama yang ditampilkan kepada pengguna, seperti teks, gambar, tautan, tabel, maupun elemen multimedia.

Kode 2.1: Struktur dasar dokumen HTML

```

6 1 <!DOCTYPE html>
7 2 <html>
8 3 <head>
9 4   <title>Contoh Halaman</title>
10 5   <meta charset="UTF-8">
11 6   <link rel="stylesheet" href="style.css">
12 7   <script src="script.js"></script>
13 8 </head>
14 9 <body>
15 10   <h1>Selamat Datang</h1>
16 11   <p>Ini adalah contoh struktur dasar HTML.</p>
17 12 </body>
18 13 </html>

```

- 21 Kode 2.1 menunjukkan struktur dasar dari HTML, baris pertama berisi deklarasi `<!DOCTYPE html>`.
- 22 Elemen `<html>` membungkus seluruh dokumen, sedangkan `<head>` berisi `<title>` sebagai judul halaman, `<meta charset="UTF-8">` untuk menetapkan karakter encoding, `<link>` untuk memanggil stylesheet eksternal, serta `<script>` untuk menyertakan berkas JavaScript. Bagian `<body>` menampilkan konten kepada pengguna, dalam contoh ini berupa judul dan paragraf.

2.3.2 Elemen Semantik dan Non-Semantik

- 27 HTML menyediakan dua jenis elemen utama, yaitu elemen semantik dan elemen non-semantik.

1. Elemen Semantik

Elemen semantik adalah elemen yang memiliki makna jelas bagi *browser* maupun pembaca, karena nama elemen menggambarkan fungsinya. Contoh elemen semantik antara lain:

- 31 • `<header>`: untuk bagian kepala suatu halaman atau bagian.
- 32 • `<nav>`: untuk kumpulan tautan navigasi.
- 33 • `<section>`: untuk sebuah bagian tematik dalam dokumen.
- 34 • `<article>`: untuk konten yang berdiri sendiri, seperti artikel berita.
- 35 • `<aside>`: untuk konten samping, seperti catatan atau iklan.
- 36 • `<footer>`: untuk bagian kaki halaman.

2. Elemen Non-Semantik

Elemen non-semantik adalah elemen yang tidak menggambarkan makna spesifik dari kontennya, melainkan digunakan untuk keperluan pemformatan atau pengelompokan. Contoh elemen non-semantik adalah:

- 41 • `<div>`: untuk pengelompokan blok konten.
- 42 • ``: untuk pengelompokan teks dalam baris.

2.3.3 Atribut Global dan Spesifik

- 44 Setiap elemen HTML dapat memiliki atribut yang berfungsi untuk memberikan informasi tambahan atau mengatur perilaku elemen tersebut. Atribut terbagi menjadi dua kategori, yaitu atribut global dan atribut spesifik.

1 Atribut global dapat digunakan pada hampir semua elemen HTML. Atribut ini bersifat umum
 2 karena tidak terikat pada fungsi tertentu dari elemen. Contoh atribut global meliputi:
 3 • **id** : identifikasi unik untuk sebuah elemen.
 4 • **class** : pengelompokan elemen dengan gaya atau fungsi yang sama.
 5 • **style** : mendefinisikan gaya inline menggunakan CSS.
 6 • **title** : menyediakan keterangan tambahan yang biasanya ditampilkan sebagai *tooltip*.
 7 Atribut spesifik adalah atribut yang hanya berlaku pada elemen tertentu sesuai dengan fungsinya.
 8 Tabel 2.1 menunjukkan beberapa contoh atribut spesifik beserta elemen tempat atribut tersebut
 9 digunakan.

Tabel 2.1: Contoh atribut spesifik pada elemen HTML

Elemen	Atribut	Keterangan
<a>	href	Menentukan alamat tujuan tautan.
	src	Menentukan lokasi berkas gambar.
<form>	action	Menentukan alamat tujuan pengiriman data formulir.
<script>	src	Menentukan sumber berkas JavaScript eksternal.
<link>	href	Menentukan lokasi stylesheet atau sumber daya terkait lainnya.

10 2.4 Web Crawling [4]

11 *Web crawling* adalah proses otomatis untuk mengambil dokumen dari web dan mengekstraksi tautan
 12 yang terkandung di dalamnya untuk penjelajahan lebih lanjut. Proses ini dijalankan oleh perangkat
 13 lunak yang disebut *crawler* atau *spider*, yang bekerja secara iteratif dengan memulai dari satu atau
 14 lebih URL awal (*seed URL*) dan mengikuti *hyperlink* yang ditemukan dalam halaman web.

15 2.4.1 Algoritma Crawling

16 *Web crawling* dapat dipandang sebagai proses eksplorasi graf, di mana simpul merepresentasikan
 17 halaman web dan sisi merepresentasikan tautan antar halaman. Untuk mengatur proses eksplorasi,
 18 *crawler* mempertahankan sebuah struktur antrean yang disebut *frontier*. Frontier adalah kumpulan
 19 URL yang telah ditemukan tetapi belum dikunjungi. URL baru yang diekstraksi dari suatu halaman
 20 ditambahkan ke frontier jika belum pernah dimasukkan sebelumnya, sedangkan URL yang sudah
 21 dikunjungi disimpan dalam sebuah himpunan terpisah untuk mencegah duplikasi.

22 Secara umum, proses *crawling* dilakukan sebagai berikut:

- 23 1. Ambil sebuah URL dari frontier sesuai strategi antrean yang digunakan.
- 24 2. Lakukan permintaan HTTP untuk mengambil dokumen dari URL tersebut.
- 25 3. Ekstrak tautan dari dokumen HTML yang diperoleh.
- 26 4.Tambahkan tautan baru ke frontier jika belum pernah dikunjungi.

27 Terdapat beberapa strategi yang dapat digunakan untuk menentukan urutan eksplorasi halaman
 28 web. Dua strategi yang umum adalah sebagai berikut:

- 29 • **Breadth-First Crawling**

30 Pada strategi ini, frontier diimplementasikan sebagai antrean FIFO (*First-In First-Out*). URL

yang pertama kali ditemukan akan dikunjungi terlebih dahulu. Pendekatan ini menghasilkan cakupan eksplorasi yang merata dan cenderung mengunjungi halaman-halaman populer lebih awal karena sifat distribusi *power-law* pada web.

- **Naïve Best-First Crawling**

Pada strategi ini, frontier diimplementasikan sebagai *priority queue*, di mana setiap URL diberi skor prioritas. Skor tersebut dapat ditentukan berdasarkan faktor topologis seperti jumlah tautan masuk, atau berdasarkan kesesuaian konten dengan suatu topik tertentu. URL dengan skor prioritas tertinggi akan dikunjungi lebih dahulu, sehingga strategi ini banyak digunakan dalam *focused crawling*.

2.4.2 Jenis-Jenis *Crawler*

Berdasarkan tujuan dan ruang lingkup *crawling*, terdapat beberapa tipe *crawler* yang umum dibahas dalam literatur:

- **Universal Crawler**

Universal crawler dirancang untuk menjelajahi bagian besar dari web tanpa pembatasan topik. Jenis ini umumnya digunakan oleh mesin pencari berskala besar, sehingga membutuhkan infrastruktur terdistribusi dan mekanisme skalabilitas tinggi untuk menangani miliaran halaman web.

- **Focused Crawler**

Focused crawler ditujukan untuk mengambil hanya halaman-halaman yang relevan terhadap suatu topik tertentu. Seleksi dilakukan dengan mengevaluasi konten atau atribut halaman sebelum mengikuti tautan. Dengan pendekatan ini, efisiensi meningkat karena *crawler* tidak mengunjungi halaman yang dianggap tidak relevan.

- **Topical Crawler**

Topical crawler merupakan variasi dari focused *crawler* yang menekankan pada keterkaitan topik antar halaman yang saling terhubung. Prinsip *topical locality* digunakan untuk memperkirakan bahwa halaman yang berdekatan dengan halaman relevan cenderung juga relevan terhadap topik yang sama. Strategi ini membantu *crawler* memilih jalur penelusuran yang lebih sesuai dengan domain yang ditargetkan.

2.4.3 Tantangan Implementasi

Pengembangan sistem *web crawling* menghadapi sejumlah tantangan teknis yang telah banyak dibahas dalam literatur. Beberapa tantangan utama tersebut adalah sebagai berikut:

- **Fetching**

Proses pengambilan konten dari halaman web harus dilakukan secara efisien. *Crawler* perlu menetapkan batas waktu (*timeout*) agar tidak menunggu terlalu lama pada halaman yang tidak responsif. Selain itu, diperlukan mekanisme pengendalian kecepatan permintaan (*rate limiting*) dengan cara menyisipkan jeda (*delay*) antar permintaan atau membatasi jumlah permintaan dalam satuan waktu. Hal ini mencegah beban berlebihan pada server tujuan serta mengurangi risiko pemblokiran alamat IP.

- **Parsing**

Konten HTML yang diperoleh dari web harus diurai (*parse*) menjadi representasi struktur

DOM untuk memungkinkan ekstraksi elemen seperti `<a href>`. Tantangan muncul karena halaman web sering kali tidak mematuhi standar HTML, misalnya memiliki tag yang tidak lengkap atau sintaks yang salah. Oleh karena itu, diperlukan parser yang toleran terhadap kesalahan dan mampu menangani berbagai *encoding* karakter. Selain itu, konten dinamis berbasis JavaScript biasanya tidak tersedia langsung dalam HTML statis sehingga dapat terlewat oleh *crawler* tradisional.

- ***Link Extraction* dan *Canonicalization***

Semua tautan yang ditemukan harus dinormalisasi agar tidak terjadi duplikasi akibat variasi penulisan URL. Normalisasi mencakup langkah-langkah seperti konversi nama host menjadi huruf kecil, penghapusan fragmen (#), parameter sesi, serta penyeragaman penggunaan *trailing slash*. Dengan demikian, *crawler* dapat memastikan setiap sumber daya diidentifikasi secara unik dan tidak mengambil halaman yang sama berulang kali.

- ***Spider Trap* dan *Infinite Loops***

Beberapa situs web berisi pola tautan yang dapat membuat *crawler* terjebak dalam perulangan tak terbatas, misalnya kalender dinamis yang menghasilkan jumlah halaman tanpa akhir atau URL dengan parameter yang terus berubah. Untuk mengatasi masalah ini, *crawler* perlu membatasi kedalaman penelusuran (*crawl depth*), menetapkan jumlah maksimum URL per domain, dan menerapkan deteksi pola berulang yang berpotensi menjadi jebakan.

- ***Page Repository***

Sistem *crawler* perlu menyimpan metadata setiap halaman yang sudah dikunjungi, termasuk URL, waktu kunjungan terakhir, status pengambilan, dan sidik jari konten (*content hash*). Penyimpanan informasi ini mencegah kunjungan ulang yang tidak diperlukan dan mendukung penerapan *scheduled recrawling* untuk memperbarui konten secara periodik.

- ***Concurrency***

Untuk meningkatkan kecepatan *crawling*, *crawler* modern menjalankan beberapa permintaan secara paralel menggunakan banyak utas (*threads*) atau proses. Akan tetapi, peningkatan jumlah permintaan simultan dapat menimbulkan risiko pembatasan atau pemblokiran oleh server target. Oleh karena itu, *crawler* memerlukan strategi sinkronisasi, antrian tugas (*task queue*), serta mekanisme pengendalian beban untuk menjaga efisiensi sekaligus menghindari gangguan pada server.

31 2.4.4 Etika dan Kepatuhan

32 Aktivitas *web crawling* tidak hanya menghadapi tantangan teknis, tetapi juga harus memperhatikan
33 aspek etika dan kepatuhan terhadap norma yang berlaku di lingkungan web. Beberapa prinsip
34 yang umum diterapkan adalah sebagai berikut:

- **`robots.txt`**

File `robots.txt` digunakan dalam protokol *Robots Exclusion Protocol* untuk menentukan bagian situs yang boleh atau tidak boleh diakses oleh *crawler*. File ini ditempatkan di direktori akar (/) dari sebuah situs web. Sebagai contoh:

```
39    User-agent: *
40    Disallow: /admin/
41    Allow: /public/
```

1 Instruksi tersebut menyatakan bahwa semua agen otomatis (ditandai dengan *) tidak diperbolehkan mengakses direktori /admin/, tetapi diperbolehkan mengakses /public/. Walaupun kepatuhan terhadap robots.txt bersifat sukarela, pengabaian aturan ini dapat dianggap sebagai pelanggaran etika.
2
3
4

5 • **User-Agent Identification**

6 Setiap permintaan HTTP memiliki *header User-Agent* yang berfungsi mengidentifikasi *crawler*.
7 Informasi ini memungkinkan administrator situs mengetahui identitas perangkat lunak yang
8 melakukan *crawling*. Sebagai praktik yang baik, *crawler* mencantumkan nama, versi, serta
9 alamat URL untuk dokumentasi atau kontak. Contoh:

10 `User-Agent: MyCrawler/1.0 (+http://example.com/crawler-info)`

11 Dengan adanya identitas ini, pemilik situs dapat memahami tujuan *crawling* dan memiliki
12 kesempatan untuk berkomunikasi dengan pengembang *crawler* bila diperlukan.

13 • **Kepatuhan terhadap Kebijakan Situs**

14 *crawler* harus menghormati kebijakan akses dan tidak mengambil konten yang bersifat pribadi,
15 rahasia, atau dilindungi hak cipta. Praktik seperti bypass autentikasi, pengambilan data
16 berbayar, atau eksplorasi celah keamanan termasuk kategori penyalahgunaan yang melanggar
17 hukum maupun etika.

18

2.5 JavaFX [5]

19 JavaFX merupakan pustaka antarmuka pengguna grafis (GUI) modern untuk bahasa pemrograman
20 Java. Pustaka ini dikembangkan sebagai penerus dari Swing dengan pendekatan yang lebih modular,
21 dukungan pemisahan logika dan tampilan, serta gaya deklaratif melalui FXML dan pengaturan
22 visual menggunakan CSS.

23 Antarmuka pengguna dalam JavaFX dibangun berdasarkan konsep *scene graph*, yaitu struktur
24 hierarkis tempat setiap elemen antarmuka direpresentasikan sebagai simpul (*Node*) dalam sebuah
25 pohon. Struktur ini diawali dari sebuah simpul akar yang dimuat ke dalam *Scene*, dan selanjutnya
26 ditampilkan pada jendela aplikasi yang disebut *Stage*.

27 Setiap *Node* dapat berupa komponen sederhana seperti *Button*, *Label*, dan *TextField*, maupun
28 kontainer tata letak seperti *VBox*, *HBox*, dan *GridPane*. Penyusunan antarmuka dalam bentuk graf
29 pohon memungkinkan pengembang untuk membangun tampilan yang konsisten, modular, dan
30 mudah dipelihara.

31

2.5.1 Siklus Hidup Aplikasi

32 Setiap aplikasi JavaFX diturunkan dari kelas abstrak `javafx.application.Application`. Kelas
33 ini mendefinisikan tiga metode utama yang membentuk siklus hidup aplikasi, yaitu:

34 1. `init()`

35 Metode ini dipanggil pertama kali sebelum antarmuka pengguna dibuat. Umumnya digunakan
36 untuk melakukan inisialisasi data awal atau membuka koneksi yang dibutuhkan.

37 2. `start(Stage primaryStage)`

38 Merupakan titik masuk utama aplikasi JavaFX. Pada tahap ini, pengembang membuat objek
39 *Scene*, mengatur elemen antarmuka pengguna, dan menampilkannya pada *Stage* utama.

1 3. **stop()**

2 Metode ini dipanggil ketika aplikasi ditutup. Biasanya digunakan untuk membersihkan sumber
3 daya, menutup koneksi, atau menyimpan data sebelum aplikasi berhenti.

4 Kode 2.2 memperlihatkan contoh struktur program JavaFX yang memuat antarmuka dari berkas
5 FXML dan menerapkan gaya visual dengan CSS.

Kode 2.2: Struktur dasar aplikasi JavaFX

```

6 1 @Override
7 2 public void start(Stage stage) throws Exception {
8 3     FXMLLoader loader = new FXMLLoader(getClass().getResource("view.fxml"));
9 4     Scene scene = new Scene(loader.load());
11 5
12 6     scene.getStylesheets().add(getClass().getResource("style.css").toExternalForm());
13 7
14 8     stage.setTitle("Contoh JavaFX");
15 9     stage.setScene(scene);
16 0     stage.setMaximized(true);
17 1     stage.show();
18 2 }
```

20 Kode pada Kode 2.2 menunjukkan bagaimana sebuah aplikasi JavaFX dimulai dari metode
21 **start**. Pertama, sebuah objek **FXMLLoader** dibuat untuk memuat berkas **view.fxml**, yang berisi
22 definisi antarmuka pengguna secara deklaratif. Hasil pemuatan berkas tersebut kemudian dibungkus
23 ke dalam sebuah objek **Scene**, sehingga seluruh elemen UI yang didefinisikan di FXML dapat
24 ditampilkan. Setelah itu, berkas **style.css** ditambahkan ke **scene** untuk mengatur gaya visual
25 menggunakan CSS. Tahap berikutnya adalah konfigurasi jendela utama aplikasi: judul jendela
26 diatur melalui **setTitle**, lalu **scene** dipasang ke dalam **stage** dengan **setScene**. Agar jendela tampil
27 penuh, **stage** diperbesar ke ukuran maksimum melalui **setMaximized(true)**. Akhirnya, aplikasi
28 ditampilkan ke layar dengan memanggil metode **show()** pada objek **stage**.

29 **2.5.2 Stage, Scene, dan Node**

30 JavaFX menyusun antarmuka pengguna berdasarkan hierarki yang dikenal dengan *scene graph*.
31 Tiga komponen utama dalam hierarki ini adalah sebagai berikut:

- 32 • **Stage**

33 Merupakan jendela utama aplikasi. JavaFX secara otomatis menyediakan sebuah **stage default**
34 yang diteruskan ke metode **start**. Di dalam **stage**, pengembang menempatkan sebuah **Scene**
35 sebagai wadah tampilan.

- 36 • **Scene**

37 Berfungsi sebagai kontainer yang memuat seluruh elemen antarmuka. Setiap **scene** me-
38 miliki satu node akar (root node), biasanya berupa komponen layout seperti **VBox**, **HBox**,
39 atau **BorderPane**. Dari simpul akar ini, elemen-elemen lain ditambahkan secara hierarkis
40 membentuk struktur pohon.

- 41 • **Node**

42 Merepresentasikan setiap elemen individual dalam antarmuka. Node dapat berupa kontrol
43 sederhana seperti **Button**, **Label**, dan **TextField**, maupun berupa kontainer yang menampung
44 node-node lain. Karena semua elemen JavaFX merupakan turunan dari kelas **Node**, pengelolaan
45 antarmuka dapat dilakukan secara konsisten.

1 2.5.3 Komponen FXML

2 JavaFX menyediakan beragam komponen antarmuka yang dapat digunakan untuk membangun
3 aplikasi interaktif. Beberapa komponen yang umum dipakai antara lain:

- **TextField**, untuk menerima input teks satu baris.
 - **TextArea**, untuk menerima input teks dalam bentuk multi-baris.
 - **ComboBox**, untuk menampilkan pilihan dalam bentuk dropdown.
 - **Button**, untuk memicu suatu aksi tertentu.
 - **Label**, untuk menampilkan teks statis maupun dinamis.
 - **TableView**, untuk menyajikan data dalam bentuk baris dan kolom.
 - **VBox**, **HBox**, dan **GridPane**, sebagai kontainer tata letak vertikal, horizontal, dan grid.

11 Komponen-komponen tersebut dapat didefinisikan secara deklaratif dengan menggunakan FXML,
12 yaitu format berbasis XML yang memisahkan struktur antarmuka dari logika aplikasi. Dalam berkas
13 FXML, sebuah kelas controller dapat dihubungkan melalui atribut `fx:controller`, sedangkan
14 komponen yang perlu diakses dari logika program diberi identitas dengan atribut `fx:id`. Selain itu,
15 event handler dapat ditentukan dengan atribut seperti `onAction`.

¹⁶ Kode [2.3](#) memperlihatkan contoh struktur antarmuka berbasis FXML yang menggunakan
¹⁷ BorderPane dan HBox sebagai kontainer tata letak.

Kode 2.3: Contoh struktur antarmuka menggunakan FXML

```
18 <BorderPane xmlns:fx="http://javafx.com/fxml"
19   fx:controller="com.example.Controller">
20
21   <top>
22     <HBox spacing="10">
23       <Label text="Input:"/>
24       <TextField fx:id="inputField"/>
25       <Button fx:id="submitButton" text="Submit" onAction="#handleSubmit"/>
26     </HBox>
27   </top>
28 </BorderPane>
```

30 Kode pada Kode 2.3 menunjukkan bahwa elemen akar adalah `BorderPane`. Pada bagian atasnya
31 (`top`) terdapat sebuah `HBox` dengan jarak antar komponen sebesar 10 piksel. Di dalam `HBox`
32 didefinisikan tiga elemen: sebuah `Label` dengan teks statis, sebuah `TextField` dengan `fx:id`
33 `inputField` agar dapat diakses dari controller, serta sebuah `Button` dengan teks “Submit” yang
34 memiliki event handler `onAction` untuk memanggil metode `handleSubmit`.

35 2.5.4 Controller, Property, dan Binding

36 Dalam arsitektur JavaFX, **controller** berperan sebagai penghubung antara antarmuka pengguna
37 yang didefinisikan melalui FXML dengan logika aplikasi. Komponen yang ada di dalam berkas
38 FXML dapat diakses dari kelas controller dengan menggunakan anotasi @FXML. Interaksi pengguna,
39 seperti menekan tombol, dapat ditangani melalui metode event handler yang didefinisikan dalam
40 controller.

41 Selain itu, JavaFX menyediakan mekanisme *property* untuk memudahkan pengelolaan data yang
42 bersifat dinamis. Property memungkinkan nilai suatu variabel dipantau sehingga ketika terjadi
43 perubahan, komponen antarmuka yang terhubung juga akan diperbarui. Salah satu implementasi
44 umum adalah **StringProperty** yang digunakan untuk menyimpan dan memantau nilai string.

45 Untuk menghubungkan property dengan komponen antarmuka, JavaFX mendukung konsep

1 *binding*. Binding dapat bersifat satu arah (*unidirectional*), di mana nilai hanya mengalir dari sumber
 2 ke target, maupun dua arah (*bidirectional*), di mana perubahan pada salah satu sisi secara otomatis
 3 memperbarui sisi lainnya.

4 Kode 2.4 memperlihatkan contoh kelas controller yang menggunakan property dan binding
 5 untuk menghubungkan **TextField** dengan **Label** melalui **StringProperty**.

Kode 2.4: Contoh controller dengan property dan binding

```

6
7 1 public class Controller {
8 2     @FXML private TextField inputField;
9 3     @FXML private Label statusLabel;
10 4
11 5     private StringProperty inputText = new SimpleStringProperty();
12 6
13 7     public void initialize() {
14 8         // Binding dua arah antara TextField dan property
15 9         inputField.textProperty().bindBidirectional(inputText);
16 0
17 1         // Binding satu arah dari property ke Label
18 2         statusLabel.textProperty().bind(inputText);
19 3     }
20 4
21 5     @FXML
22 6     public void handleSubmit() {
23 7         inputText.set("Input diterima: " + inputText.get());
24 8     }
25 9 }
```

27 Kode pada Kode 2.4 menunjukkan bagaimana controller bekerja dengan property dan bin-
 28 ding. Pertama, komponen **TextField** dan **Label** dihubungkan ke kelas controller melalui anotasi
 29 **@FXML**. Sebuah property berupa **StringProperty** dibuat dengan nama **inputText**. Pada metode
 30 **initialize()**, property ini diikat secara dua arah dengan teks pada **TextField**, sehingga setiap
 31 perubahan pada salah satunya langsung tercermin pada yang lain. Selanjutnya, teks pada **Label**
 32 diikat secara satu arah dengan property, sehingga nilai label selalu sesuai dengan isi property.
 33 Ketika tombol submit ditekan, metode **handleSubmit()** dijalankan, yang akan memperbarui nilai
 34 property menjadi string baru. Perubahan tersebut secara otomatis ditampilkan di **Label** karena
 35 adanya mekanisme binding.

36 2.6 Jsoup

37 Jsoup adalah sebuah pustaka Java yang dikembangkan oleh Jonathan Hedley pada tahun 2009 dan
 38 dipublikasikan sebagai proyek sumber terbuka dengan lisensi MIT. Pustaka ini berfungsi sebagai
 39 **HTML parser** yang digunakan untuk memproses dokumen HTML maupun XML sehingga dapat
 40 diolah dalam bentuk struktur yang lebih teratur.

41 Dokumen yang diproses dengan Jsoup dapat berasal dari tiga sumber utama, yaitu halaman
 42 web yang diakses melalui URL, file yang tersimpan secara lokal, dan string yang berisi kode
 43 HTML. Setelah dokumen diperoleh, Jsoup melakukan *parsing* berdasarkan spesifikasi HTML5
 44 untuk menghasilkan representasi dalam bentuk *Document Object Model* (DOM) yang konsisten
 45 dengan hasil pemrosesan browser modern.

46 Selain menghasilkan representasi DOM, Jsoup juga dirancang agar toleran terhadap dokumen
 47 HTML yang tidak valid atau tidak terstruktur. Dengan karakteristik ini, Jsoup tetap mampu
 48 membentuk pohon hasil *parsing* yang dapat diproses lebih lanjut. Pustaka ini mendukung keluaran
 49 kembali dalam bentuk HTML yang telah ditata (*tidy HTML*), sehingga dokumen hasil pemrosesan
 50 menjadi lebih rapi dan siap digunakan pada tahap berikutnya.

1 2.6.1 Kelas Utama

2 Beberapa kelas utama yang disediakan oleh Jsoup dalam pengolahan dokumen HTML adalah
3 sebagai berikut:

4 a. **Jsoup**

5 Kelas ini merupakan titik masuk utama untuk menggunakan pustaka Jsoup. Seluruh *method*
6 yang disediakan bersifat *static* dan digunakan untuk memulai proses pengambilan maupun
7 *parsing* dokumen.

- 8 • `connect(String url)`

9 Membuat sebuah koneksi HTTP ke alamat web yang ditentukan oleh parameter URL,
10 menghasilkan objek `Connection`.

- 11 • `parse(String html)`

12 Melakukan *parsing* terhadap string HTML dan menghasilkan sebuah objek `Document`.

- 13 • `parse(File file, String charsetName)`

14 Memproses file HTML lokal dengan karakter encoding tertentu, menghasilkan objek
15 `Document`.

16 b. **Connection**

17 Antarmuka yang merepresentasikan konfigurasi sebuah koneksi HTTP sebelum dijalankan.
18 Melalui `Connection`, berbagai properti *request* dapat diatur.

- 19 • `userAgent(String ua)`

20 Menentukan nilai *User-Agent* yang dikirimkan pada *request* HTTP.

- 21 • `timeout(int millis)`

22 Menentukan batas waktu koneksi dalam satuan milidetik.

- 23 • `method(Connection.Method m)`

24 Menetapkan HTTP *method* yang akan digunakan, seperti GET, POST, atau HEAD.

- 25 • `get()`

26 Mengeksekusi HTTP *request* dengan *method* GET dan menghasilkan objek `Document`.

- 27 • `execute()`

28 Mengeksekusi *request* dan mengembalikan objek `Connection.Response` yang memuat
29 informasi lengkap hasil *request*.

30 c. **Connection.Response**

31 Antarmuka yang merepresentasikan *response* HTTP yang diterima setelah *request* dijalankan.

- 32 • `statusCode()`

33 Mengembalikan HTTP *status code* dari *response*, seperti 200 atau 404.

- 34 • `headers()`

35 Mengembalikan seluruh *header* dari *response* dalam bentuk pasangan nama–nilai.

- 36 • `cookies()`

37 Mengembalikan seluruh *cookie* yang diterima dari server.

- 38 • `body()`

39 Mengembalikan isi *response body* dalam bentuk string.

- 40 • `parse()`

41 Memproses isi *response body* menjadi objek `Document`.

42 d. **Document**

1 Kelas yang merepresentasikan sebuah dokumen HTML utuh. `Document` merupakan turunan
2 dari `Element`, sehingga mewarisi banyak *method* darinya.

- 3 • `title()`

4 Mengembalikan nilai judul halaman HTML dari elemen `<title>`.

- 5 • `body()`

6 Mengembalikan elemen `<body>` dari dokumen.

- 7 • `select(String cssQuery)`

8 Mencari elemen-elemen yang sesuai dengan CSS *selector* dan mengembalikan objek
9 `Elements`.

- 10 • `baseUri()`

11 Mengembalikan URI dasar dari dokumen yang digunakan untuk menyelesaikan URL
12 relatif.

13 e. **Element**

14 Kelas yang merepresentasikan sebuah elemen tunggal dalam struktur HTML, seperti `<a>`,
15 `<p>`, atau `<div>`.

- 16 • `attr(String key)`

17 Membaca atau mengubah nilai atribut dari sebuah elemen.

- 18 • `text()`

19 Mengembalikan isi teks dari elemen tanpa menyertakan *markup* HTML.

- 20 • `html()`

21 Mengembalikan isi HTML dari elemen, termasuk *markup* di dalamnya.

- 22 • `absUrl(String key)`

23 Mengembalikan URL absolut dari sebuah atribut, seperti `href` atau `src`, berdasarkan
24 URI dasar dokumen.

25 f. **Elements**

26 Kelas yang merepresentasikan sekumpulan objek `Element`. Biasanya dihasilkan oleh pemang-
27 gilan *method* `select()`.

- 28 • `size()`

29 Mengembalikan jumlah elemen dalam koleksi.

- 30 • `get(int index)`

31 Mengambil elemen pada posisi tertentu di dalam koleksi.

- 32 • `eachText()`

33 Mengembalikan daftar teks dari seluruh elemen dalam koleksi.

34 g. **Parser**

35 Kelas yang digunakan untuk melakukan *parsing* terhadap dokumen HTML atau XML.

- 36 • `htmlParser()`

37 Menghasilkan parser HTML sesuai spesifikasi HTML5.

- 38 • `xmlParser()`

39 Menghasilkan parser XML untuk memproses dokumen XML.

- 40 • `parse(String html, String baseUri)`

41 Memproses string HTML dengan URI dasar yang ditentukan, menghasilkan objek
42 `Document`.

1 h. Selector

2 Kelas yang digunakan secara internal untuk mendukung mekanisme CSS selector pada Jsoup.

3 • `select(String cssQuery, Element root)`

4 Mencari elemen-elemen yang sesuai dengan *query* CSS dari suatu elemen *root*.

5 • `selectFirst(String cssQuery, Element root)`

6 Mengembalikan elemen pertama yang sesuai dengan *query* CSS dari suatu elemen *root*.

7 • `evaluate(String cssQuery, Element root)`

8 Mengevaluasi *query* CSS terhadap elemen *root* untuk menghasilkan koleksi elemen yang
9 sesuai.

10 Selain kelas-kelas utama tersebut, Jsoup juga mendefinisikan beberapa kelas *exception*, antara lain

11 `HttpStatusException`, yang dilemparkan ketika server mengembalikan HTTP *status code* selain

12 2xx, serta `UnsupportedMimeTypeException`, yang muncul apabila dokumen memiliki tipe konten

13 yang tidak dapat diproses sebagai HTML.

14 **2.6.2 Contoh Kode Program**

15 Kode 2.5 menunjukkan contoh penggunaan Jsoup untuk mengambil sebuah halaman web, memprosesnya menjadi objek `Document`, dan mengekstraksi sejumlah informasi dari dokumen tersebut.

16 Contoh ini menggambarkan bagaimana kelas-kelas utama yang telah dijelaskan sebelumnya, yaitu

17 Jsoup, Connection, Connection.Response, Document, Elements, dan Element, digunakan secara

18 bersama-sama dalam sebuah program nyata. Hasil eksekusi dari kode ini berupa informasi judul

19 halaman, URI dasar dokumen, serta daftar tautan beserta teks, URL absolut, dan atribut lain yang

20 terkait.

Kode 2.5: Contoh Penggunaan Jsoup

```

22
23 1 import org.jsoup.Connection;
24 2 import org.jsoup.Jsoup;
25 3 import org.jsoup.nodes.Document;
26 4 import org.jsoup.nodes.Element;
27 5 import org.jsoup.select.Elements;
28 6
29 7 public class JsoupExample {
30 8     public static void main(String[] args) throws Exception {
31 9         // Membuat koneksi dengan pengaturan user-agent dan timeout
32 0         Connection connection = Jsoup.connect("https://www.example.com")
33 1             .userAgent("BrokenLinkChecker_1.0")
34 2             .timeout(5000);
35 3
36 4         // Mengeksekusi request dan memperoleh response
37 5         Connection.Response response = connection.execute();
38 6
39 7         // Menampilkan status code dan content type
40 8         System.out.println("Status_Code:_" + response.statusCode());
41 9         System.out.println("Content-Type:_" + response.headers().get("Content-Type"));
42 0
43 1         // Memproses response menjadi Document
44 2         Document doc = response.parse();
45 3
46 4         System.out.println("Title:_" + doc.title());
47 5         System.out.println("Base_URI:_" + doc.baseUri());
48 6
49 7         // Menyeleksi semua elemen <a> yang memiliki atribut href
50 8         Elements links = doc.select("a[href]");
51 9
52 0         for (Element link : links) {
53 1             System.out.println("Teks:_" + link.text());
54 2             System.out.println("Href:_" + link.attr("href"));
55 3             System.out.println("AbsURL:_" + link.absUrl("href"));
56 4         }
57 5     }
58 6 }
59 7

```

1 Alur dari Kode 2.5 dimulai dengan pembuatan objek `Connection` melalui pemanggilan metode
2 `Jsoup.connect()`, kemudian ditetapkan nilai *User-Agent* dengan `userAgent()` dan batas waktu
3 koneksi dengan `timeout()`. Permintaan dieksekusi melalui `execute()` untuk menghasilkan objek
4 `Connection.Response`, dari mana dapat diperoleh kode status dengan `statusCode()` dan *header*
5 tertentu melalui `headers()`. Isi respons kemudian diproses menjadi objek `Document` menggunakan
6 `parse()`, yang selanjutnya menyediakan informasi judul halaman dengan `title()` dan URI dasar
7 dengan `baseUri()`. Setelah itu, method `select()` digunakan untuk memilih seluruh elemen `<a>`
8 yang memiliki atribut `href`, menghasilkan koleksi `Elements`. Koleksi ini diiterasi, dan untuk setiap
9 `Element` diperoleh teks melalui `text()`, nilai atribut melalui `attr("href")`, serta URL absolut
10 melalui `absUrl("href")`. Dengan demikian, kode ini tidak hanya menampilkan daftar tautan,
11 tetapi juga memperlihatkan cara menggunakan beberapa method penting yang telah dijelaskan
12 sebelumnya.

13 2.7 Java HTTP Client API

14 Java HTTP Client API secara garis besar adalah sebuah API dari Java yang digunakan untuk
15 mengirim *request* dan menerima *response* melalui protokol HTTP. API ini pertama kali diperkenalkan
16 pada Java 9 dengan nama `jdk.incubator.httpclient` sebagai *incubating module*, yaitu modul
17 percobaan yang digunakan untuk memperkenalkan API baru sebelum akhirnya ditetapkan sebagai
18 bagian resmi dari Java 11. Java HTTP Client API mendukung penggunaan protokol HTTP/1.1 dan
19 HTTP/2. Secara bawaan, pemilihan versi protokol dilakukan secara otomatis, di mana `HttpClient`
20 akan mencoba menggunakan HTTP/2 terlebih dahulu dan melakukan *fallback* ke HTTP/1.1 apabila
21 server tidak mendukung HTTP/2. Selain mekanisme bawaan tersebut, pengembang juga dapat
22 menetapkan versi protokol secara eksplisit melalui metode yang tersedia pada kelas `HttpClient`.

23 Selain mendukung dua versi protokol HTTP, API ini juga menyediakan dua model komunikasi,
24 yaitu sinkron dan asinkron. Komunikasi sinkron berarti eksekusi program akan menunggu hingga
25 *response* (*response*) dari server diterima sepenuhnya sebelum melanjutkan instruksi berikutnya.
26 Sebaliknya, komunikasi asinkron menggunakan kelas `CompletableFuture`, yang memungkinkan hasil
27 komputasi diperoleh di masa mendatang tanpa harus menunggu proses selesai. Objek `HttpClient`
28 dalam API ini juga memiliki karakteristik penting untuk penggunaan di lingkungan *multithreaded*.
29 Objek ini bersifat *immutable*, artinya konfigurasi tidak dapat diubah setelah dibuat, serta bersifat
30 *thread-safe*, artinya dapat diakses secara bersamaan oleh beberapa *thread* tanpa menimbulkan *race*
31 *condition*.

32 2.7.1 HttpClient

33 Kelas `HttpClient` merupakan komponen inti dalam Java HTTP Client API yang digunakan untuk
34 mengirim *request* HTTP dan menerima *response* dari server. Objek dari kelas ini dapat dibuat
35 dengan menggunakan *static method* `newHttpClient` dengan konfigurasi bawaan yang siap pakai,
36 atau bisa menggunakan *static method* `newBuilder` apabila ingin membuat objek dengan konfigurasi
37 yang disesuaikan.

38 Berikut ini adalah beberapa *method* untuk menetapkan konfigurasi, yang tersedia pada objek
39 *builder* `newBuilder`:

1 • **version**

2 Method ini berfungsi untuk menentukan versi protokol HTTP yang akan digunakan, yaitu
3 HTTP_1_1 atau HTTP_2. Nilai yang diberikan berasal dari enum `HttpClient.Version`. Dengan
4 pengaturan ini, klien dapat diarahkan untuk selalu memakai versi tertentu atau menyesuaikan
5 melalui negosiasi otomatis dengan server. Pengaturan versi yang jelas membantu menjaga
6 konsistensi komunikasi dan mencegah terjadinya ketidakcocokan protokol antara klien dan
7 server.

8 • **connectTimeout**

9 Method ini menerima sebuah objek `Duration` yang menyatakan batas waktu maksimal dalam
10 membangun koneksi ke server. Jika dalam jangka waktu tersebut koneksi tidak berhasil
11 dibuat, maka proses akan dihentikan dan menghasilkan *exception*. Dengan demikian, aplikasi
12 tidak akan menggantung terlalu lama ketika mencoba mengakses server yang tidak responsif.

13 • **cookieHandler**

14 Method ini digunakan untuk menetapkan objek `CookieHandler` yang akan menangani mana-
15 jemen *cookie* selama komunikasi antara *client* dan *server*. Ketika sebuah *request* atau *response*
16 berisi *cookie*, objek `CookieHandler` akan menyimpan, memperbarui, dan mengirimkan kembali
17 *cookie* tersebut sesuai aturan yang ditentukan. Dengan pengaturan ini, *client* dapat
18 mempertahankan sesi secara otomatis, mirip dengan perilaku *browser* yang menyimpan *cookie*
19 dan menggunakan kembali pada *request* berikutnya.

20 • **followRedirects**

21 Method ini mengatur kebijakan ketika server memberikan instruksi *redirect*. Parameter
22 yang digunakan adalah nilai dari enum `HttpClient.Redirect`, seperti **ALWAYS** untuk selalu
23 mengikuti *redirect*, **NEVER** untuk tidak pernah mengikuti, dan **NORMAL** untuk selalu mengikuti
24 *redirect*, kecuali dari URL dengan *scheme* HTTPS ke HTTP.

25 • **authenticator**

26 Method ini digunakan untuk menetapkan sebuah objek `Authenticator` yang akan dipanggil
27 ketika server meminta autentikasi. Misalnya, saat server meminta kredensial melalui HTTP
28 Basic Authentication, objek ini dapat menyediakan username dan password. Dengan demikian,
29 akses ke sumber daya yang dilindungi dapat dilakukan secara otomatis tanpa perlu intervensi
30 manual.

31 • **build**

32 Setelah seluruh konfigurasi ditentukan, *method* ini dipanggil untuk menghasilkan sebuah
33 objek `HttpClient` yang siap digunakan. Objek ini bersifat *immutable*, artinya setelah diban-
34 gun, konfigurasi tidak dapat diubah lagi, sehingga memastikan konsistensi perilaku selama
35 penggunaan.

36 Selain *static method*, kelas `HttpClient` juga menyediakan sejumlah *instance method* yang digu-
37 nakan secara langsung pada objek hasil *build*. Salah satu yang paling penting adalah `send`, yaitu
38 *method* yang menjalankan sebuah *request* HTTP secara sinkron. *Method* ini menerima parameter
39 berupa objek `HttpRequest` dan `HttpResponse.BodyHandler`, kemudian mengeksekusi *request* hing-
40 ga selesai dan mengembalikan objek `HttpResponse` yang berisi *status code*, *header*, serta isi *response*
41 *body*. Selain itu, terdapat juga *method* `sendAsync` yang bekerja secara asinkron. Sama seperti
42 `send`, *method* ini menerima objek `HttpRequest` dan `HttpResponse.BodyHandler`, namun alih-alih

- 1 langsung menghasilkan `HttpResponse`, `method` ini mengembalikan sebuah `CompletableFuture`
- 2 yang merepresentasikan hasil `response` yang akan tersedia di kemudian waktu.

3 2.7.2 HttpRequest

4 Kelas `HttpRequest` merepresentasikan sebuah *request* HTTP yang akan dikirim menggunakan
5 `HttpClient`. Objek dari kelas ini bersifat *immutable*, sehingga setiap konfigurasi harus diten-
6 tukan terlebih dahulu melalui `HttpRequest.Builder` sebelum *request* dibangun. Setelah dibuat,
7 konfigurasi sebuah `HttpRequest` tidak dapat diubah lagi.

8 Untuk membangun sebuah `HttpRequest`, dapat digunakan *static method* `newBuilder` yang
9 menghasilkan objek `HttpRequest.Builder`. Pada *builder* ini tersedia sejumlah *method* yang
10 digunakan untuk menetapkan konfigurasi permintaan:

- 11 • **uri**

12 *Method* ini digunakan untuk menentukan alamat tujuan dari *request* HTTP dalam bentuk
13 objek `java.net.URI`. URI bersifat wajib karena menentukan ke mana *request* akan dikirim.

- 14 • **header** dan **headers**

15 *Method* `header` menetapkan satu pasangan *name-value* sebagai *header*, sedangkan `headers`
16 digunakan untuk menetapkan beberapa pasangan sekaligus. Header merupakan informasi
17 tambahan yang menyertai permintaan, seperti `Content-Type` atau `Authorization`.

- 18 • **timeout**

19 *Method* ini menerima parameter berupa objek `Duration` untuk menentukan batas waktu
20 maksimal pemrosesan permintaan. Jika respons tidak diterima dalam waktu yang ditetapkan,
21 maka akan terjadi *timeout*.

- 22 • **version**

23 *Method* ini digunakan untuk menentukan versi protokol HTTP yang akan digunakan pada
24 *request* tertentu. Nilai yang diberikan berupa enum `HttpClient.Version`.

- 25 • **expectContinue**

26 *Method* ini mengatur apakah *request* menggunakan mekanisme *Expect: 100-continue*. Jika
27 diset true, client akan mengirim *header* terlebih dahulu dan menunggu konfirmasi dari server
28 sebelum mengirim isi *request body*. Mekanisme ini berguna untuk *request* dengan *request body*
29 berukuran besar.

- 30 • **GET, POST, PUT, DELETE, dan method**

31 *Methods* ini menentukan jenis operasi HTTP yang akan dilakukan. GET dan DELETE tidak
32 menyertakan *request body*, sementara POST dan PUT membutuhkan objek `BodyPublisher`
33 untuk mendefinisikan data yang akan dikirim melalui *request body*. *Method* generik `method`
34 memungkinkan penentuan jenis operasi lain seperti PATCH dan HEAD.

- 35 • **build**

36 Setelah semua konfigurasi ditetapkan, `method` ini dipanggil untuk menghasilkan sebuah objek
37 `HttpRequest` yang siap digunakan oleh `HttpClient`.

38 2.7.3 HttpResponse

39 Kelas `HttpResponse` merepresentasikan *response* yang diterima setelah sebuah *request* dieksekusi
40 oleh `HttpClient`. Kelas ini menggunakan parameter generik `<T>` yang menunjukkan tipe data dari

- 1 isi *response body*. Nilai generik ini ditentukan oleh `HttpResponse.BodyHandler` yang digunakan
2 saat mengirim permintaan.
- 3 Beberapa *method* penting yang tersedia pada `HttpResponse` antara lain:
- 4 • **statusCode**
5 Mengembalikan nilai berupa bilangan bulat yang merepresentasikan HTTP *status code* dari
6 *response*, misalnya 200 untuk *OK* atau 404 untuk *Not Found*.
 - 7 • **headers**
8 Mengembalikan objek `HttpHeaders` yang berisi seluruh *header* dari *response*. Setiap *header*
9 dapat diakses berdasarkan nama dan dapat memiliki lebih dari satu nilai.
 - 10 • **body**
11 Mengembalikan isi *response body* dengan tipe data sesuai parameter generik <T>. Contohnya,
12 jika menggunakan `BodyHandler<String>`, maka *response body* akan dikembalikan dalam
13 bentuk string.
 - 14 • **previousResponse**
15 Mengembalikan objek `Optional<HttpResponse<T>` yang berisi *response* sebelumnya apabila
16 terjadi *redirect*. Jika tidak ada, nilai yang dikembalikan adalah kosong.
 - 17 • **sslSession**
18 Mengembalikan informasi sesi SSL dalam bentuk `Optional<SSLSession>` jika koneksi dilala-
19 kukan melalui protokol HTTPS.
 - 20 • **uri**
21 Mengembalikan objek `URI` yang merepresentasikan alamat tujuan akhir dari permintaan,
22 termasuk setelah terjadi *redirect*.
 - 23 • **version**
24 Mengembalikan versi protokol HTTP yang digunakan pada komunikasi, berupa nilai dari
25 enum `HttpClient.Version`.
 - 26 • **request**
27 Mengembalikan objek `HttpRequest` yang digunakan untuk menghasilkan *response* ini.

28 2.7.4 `HttpHeaders`

29 Kelas `HttpHeaders` merepresentasikan kumpulan *header* yang terdapat pada *request* maupun
30 *response* HTTP. Objek dari kelas ini bersifat *immutable*, sehingga nilai yang tersimpan tidak dapat
31 diubah setelah dibuat. Setiap *header* dapat memiliki lebih dari satu nilai, dan semua nilai disimpan
32 dalam struktur yang mempertahankan urutan kemunculannya.

33 Beberapa *method* penting yang tersedia pada `HttpHeaders` antara lain:

- 34 • **firstValue**
35 Mengembalikan nilai pertama dari *header* dengan nama tertentu dalam bentuk *string*. *Method*
36 ini berguna ketika hanya satu nilai yang relevan dari sebuah *header* yang mungkin memiliki
37 banyak nilai.
- 38 • **allValues**
39 Mengembalikan seluruh nilai dari *header* dengan nama tertentu dalam bentuk daftar. *Method*
40 ini digunakan ketika sebuah *header* dapat berisi lebih dari satu nilai.
- 41 • **map**

1 Mengembalikan seluruh isi *header* dalam bentuk struktur `Map<String, List<String>`, dengan setiap kunci berupa nama *header* dan nilainya berupa daftar nilai terkait. Dengan cara ini, semua *header* dan nilainya dapat diakses secara langsung.

4 2.7.5 Contoh Kode Program

5 Kode 2.6 menunjukkan contoh penggunaan Java HTTP Client API untuk melakukan permintaan
6 HTTP ke sebuah alamat web dan menampilkan hasil tanggapannya. Kode ini ditulis untuk
7 menggambarkan bagaimana kelas-kelas utama yang telah dijelaskan sebelumnya, yaitu `HttpClient`,
8 `HttpRequest`, `HttpResponse`, dan `HttpHeaders`, digunakan secara bersama-sama dalam sebuah
9 program nyata. Hasil eksekusi dari kode ini berupa informasi *status code*, daftar *header*, serta isi
10 *response body* yang diterima dari server tujuan.

Kode 2.6: Contoh penggunaan Java HTTP Client API

```
11 public class HttpClientExample {
12     public static void main(String[] args) throws Exception {
13         // Membangun HttpClient dengan konfigurasi yang relevan
14         HttpClient client = HttpClient.newBuilder()
15             .version(HttpClient.Version.HTTP_2)
16             .connectTimeout(Duration.ofSeconds(5))
17             .followRedirects(HttpClient.Redirect.NORMAL)
18             .build();
19
20         // Menyusun HttpRequest: URI, header, timeout, dan method GET
21         HttpRequest request = HttpRequest.newBuilder(URI.create("https://informatika.unpar.ac.id"))
22             .header("User-Agent", "BrokenLinkChecker_1.0")
23             .timeout(Duration.ofSeconds(10))
24             .GET()
25             .build();
26
27         HttpResponse<String> response = client.send(
28             request, HttpResponse.BodyHandlers.ofString()
29         );
30
31         System.out.println("Status_Code:_" + response.statusCode());
32
33         HttpHeaders headers = response.headers();
34         Map<String, List<String>> headerMap = headers.map();
35         for (Map.Entry<String, List<String>> e : headerMap.entrySet()) {
36             String name = e.getKey();
37             String values = String.join(", ", e.getValue());
38             System.out.println(name + ":_" + values);
39         }
40
41         System.out.println("Body_____:_" + response.body());
42     }
43 }
44 }
```

46 Alur dari Kode 2.6 dimulai dengan pembuatan objek `HttpClient` melalui `newBuilder()` dan pe-
47 netapan beberapa konfigurasi, yaitu versi protokol HTTP melalui `version()`, batas waktu koneksi de-
48 ngan `connectTimeout()`, serta kebijakan *redirect* menggunakan `followRedirects()`. Setelah itu di-
49 panggil `build()` untuk menghasilkan objek `HttpClient`. Berikutnya, sebuah `HttpRequest` dibangun
50 dengan `HttpRequest.newBuilder(URI)` untuk menentukan alamat tujuan, kemudian ditambahkan
51 informasi tambahan melalui `header()`, ditetapkan batas waktu menggunakan `timeout()`, dan jenis
52 operasi HTTP dipilih dengan `GET()`, lalu diselesaikan dengan `build()`. Permintaan tersebut dikirim
53 menggunakan `HttpClient.send()` dengan parameter `HttpResponse.BodyHandlers.ofString()`,
54 sehingga isi *response body* diproses menjadi *string*. Hasil eksekusi berupa objek `HttpResponse` yang
55 menyediakan *status code*, kumpulan *header*, serta isi *response body*. Daftar *header* diperoleh dari
56 `HttpHeaders.map()`, kemudian ditampilkan seluruhnya dalam bentuk pasangan *name-value*, diikuti
57 dengan pencetakan isi *response body*.

1

BAB 3

2

ANALISIS

3 Bab ini membahas analisis yang menjadi dasar perancangan dan pengembangan aplikasi desktop
4 pemeriksa tautan rusak pada situs web. Analisis mencakup permasalahan utama yang melatarbelakangi penelitian, tinjauan terhadap sistem serupa, serta perumusan kebutuhan sistem baik fungsional
5 maupun non-fungsional. Selain itu, dibahas pula analisis terhadap teknologi yang digunakan untuk
6 mendukung implementasi aplikasi.
7

8 3.1 Analisis Masalah

9 3.1.1 Tautan Rusak

10 Analisis mengenai tautan rusak pada penelitian ini didasarkan pada teori URI yang dijelaskan pada
11 Subbab 2.2 serta teori HTTP pada Subbab 2.1. Tujuan utamanya adalah memahami faktor-faktor
12 yang membuat tautan gagal diakses. Dengan melihat struktur URL, aturan sintaks, mekanisme
13 resolusi, hingga kode status HTTP, kita dapat memahami dengan lebih jelas bagaimana tautan
14 dinyatakan rusak dan apa saja indikator yang bisa digunakan untuk mengenalinya.

15 Struktur URL

16 Pada Subbab 2.2.1 dijelaskan bahwa URL memiliki sejumlah komponen, mulai dari *scheme*, *host*,
17 *port*, *path*, *query*, hingga *fragment*. Dari sisi analisis, tidak semua bagian berpengaruh langsung
18 pada keabsahan tautan. Komponen yang paling menentukan adalah *scheme*, *host*, *port*, *path*, dan
19 *query*, sedangkan *fragment* hanya diproses di sisi klien.

20 Beberapa contoh permasalahan yang bisa muncul:

- 21 • **Scheme** yang salah membuat agen pengguna tidak bisa membangun koneksi.
 - 22 • **Host** yang keliru, misalnya domain salah ketik atau tidak terdaftar, menyebabkan tautan
23 tidak ditemukan.
 - 24 • **Port** yang tidak sesuai membuat layanan yang dituju tidak dapat diakses.
 - 25 • **Path** yang tidak valid biasanya mengembalikan respons 404 Not Found.
 - 26 • **Query** yang salah dapat membuat server gagal menampilkan hasil yang diinginkan.
- 27 Dari sini terlihat bahwa kesalahan pada salah satu komponen utama sudah cukup untuk membuat
28 tautan tidak berfungsi.

1 Validitas Sintaks dan Encoding

2 Aturan penggunaan karakter dalam URI dan mekanisme *percent-encoding* dibahas pada Subbab 2.2.2.
3 Dalam praktiknya, banyak tautan yang rusak bukan karena salah struktur, tetapi karena melanggar
4 aturan sintaks atau *encoding*. Contoh yang sering terjadi adalah penggunaan karakter *reserved*
5 di luar konteks, adanya spasi atau karakter non-ASCII yang tidak dikodekan, atau penulisan
6 *percent-encoding* dengan format yang salah.

7 Masalah lain muncul dari normalisasi, sebagaimana dijelaskan pada Subbab ???. Misalnya *host*
8 ditulis dengan huruf besar, port default tetap dicantumkan, atau path masih berisi segmen . dan
9 ... Situasi semacam ini membuat URL ambigu. Artinya, meskipun sebuah URL terlihat benar,
10 kesalahan kecil dalam sintaks atau encoding bisa membuatnya dianggap rusak.

11 URL Relatif

12 Pada Subbab 2.2.1 dijelaskan bahwa URL relatif harus digabungkan dengan *base URI* untuk
13 membentuk URL absolut. Masalah muncul ketika *base URI* tidak jelas atau proses resolusinya
14 tidak berjalan dengan benar. Hasilnya, URL absolut yang terbentuk tidak menunjuk ke sumber
15 daya yang diinginkan.

16 Hal serupa bisa terjadi bila *dot-segments* tidak dihapus. Akibatnya, jalur yang terbentuk tidak
17 sesuai. Dalam kondisi ini, tautan yang seharusnya valid sebagai URL relatif menjadi tidak dapat
18 digunakan setelah diubah menjadi absolut.

19 Komunikasi HTTP

20 HTTP sebagai protokol komunikasi dibahas pada Subbab 2.1. Walaupun sebuah URL valid secara
21 struktur, tautan tetap bisa gagal diakses bila komunikasi HTTP tidak berhasil. Permasalahan ini
22 biasanya muncul pada tahap koneksi antara klien dan server.

23 Beberapa kasus yang sering ditemui adalah:

- **Host tidak ditemukan**, misalnya ketika domain gagal dipetakan oleh DNS.
- **Koneksi ditolak**, ketika server tidak menyediakan layanan pada port yang diminta.
- **Timeout**, ketika server tidak memberikan respons dalam batas waktu yang ditentukan.

27 Dengan kata lain, validitas URL belum cukup. Tautan juga perlu diuji pada tahap komunikasi,
28 karena kegagalan di sini tetap membuatnya rusak.

29 Kode Status HTTP

30 Pada Subbab 2.1.2 dijelaskan bahwa setiap respons HTTP memiliki kode status sebagai penanda
31 hasil permintaan. Dalam analisis tautan rusak, kode inilah yang biasanya menjadi acuan utama.

32 Beberapa kategori kode status yang sering digunakan adalah:

- **4xx Client Error**. Kategori ini menandakan kesalahan di sisi klien. Contoh yang paling
34 sering ditemui adalah 404 Not Found. Ada juga 410 Gone yang berarti sumber daya sudah
35 dihapus secara permanen, serta 403 Forbidden ketika akses ditolak.
- **5xx Server Error**. Kategori ini menunjukkan kegagalan di sisi server. Misalnya 500
37 Internal Server Error yang menandakan adanya masalah umum pada server, atau 503
38 Service Unavailable ketika layanan tidak tersedia untuk sementara.

- 1 • **Redirect loop.** Kondisi ini muncul ketika server berulang kali mengarahkan ke lokasi lain
2 dengan kode 3xx tanpa pernah benar-benar sampai ke sumber daya tujuan.
3 Dari sini dapat dipahami bahwa kode status HTTP bukan hanya informasi teknis, tetapi juga
4 indikator praktis untuk menentukan apakah sebuah tautan masih berfungsi atau sudah masuk
5 kategori rusak.

6 3.1.2 *Web Crawling*

7 Analisis pada bagian ini merujuk pada teori mengenai *web crawling* yang dibahas pada Subbab 2.4
8 serta teori HTML pada Subbab 2.3. Tujuan utamanya adalah mengidentifikasi permasalahan yang
9 muncul dalam proses *crawling* halaman web sebagai dasar pemeriksaan tautan rusak. Permasalahan
10 tersebut mencakup strategi *crawling*, jenis *crawler* yang digunakan, tantangan teknis yang dihadapi,
11 serta keterkaitannya dengan struktur HTML sebagai sumber tautan.

12 **Strategi crawling**

13 Pada Subbab 2.4 dijelaskan bahwa strategi *crawling* dapat dilakukan dengan *breadth-first crawling*
14 maupun *naïve best-first crawling*. Keduanya memiliki karakteristik yang berbeda.

- 15 • **Breadth-first crawling** menyapu halaman secara merata dari titik awal. Strategi ini cocok
16 digunakan untuk aplikasi pemeriksa tautan, karena setiap halaman pada host yang sama
17 dianggap memiliki tingkat kepentingan yang setara. Dengan cara ini, cakupan situs dapat
18 diperoleh lebih menyeluruh tanpa harus memikirkan bobot prioritas antar tautan.
- 19 • **Naïve best-first crawling** menggunakan *priority queue* berbasis skor untuk menentukan
20 halaman yang akan diambil lebih dahulu. Strategi ini bermanfaat bila ada kriteria relevansi
21 yang jelas, misalnya hanya ingin menekankan halaman dengan potensi informasi lebih tinggi.
22 Namun, dalam konteks aplikasi pemeriksa tautan rusak, pendekatan ini tidak relevan. Semua
23 halaman pada satu host dianggap sama penting, sehingga tidak ada dasar untuk memberikan
24 skor prioritas.

25 Dengan demikian, strategi yang logis untuk aplikasi ini adalah *breadth-first crawling*.

26 **Jenis Crawler**

27 Subbab 2.4 juga membahas jenis-jenis *crawler*, seperti *universal crawler*, *focused crawler*, dan *topical crawler*.

- 29 • **Universal crawler** mencoba merayapi seluruh bagian web dalam skala luas. Jenis ini tidak
30 sesuai untuk aplikasi pemeriksa tautan karena lingkupnya terlalu besar dan tidak efisien.
- 31 • **Focused crawler** membatasi diri pada kriteria tertentu. Pendekatan ini relevan karena
32 aplikasi ini hanya perlu merayapi halaman dengan host yang sama. Dengan cara ini, proses
33 *crawling* lebih terarah dan sumber daya tidak terbuang pada tautan eksternal.
- 34 • **Topical crawler** memfokuskan *crawling* pada topik tertentu. Jenis ini biasanya digunakan
35 untuk pengumpulan konten tematik. Untuk pemeriksa tautan, pendekatan ini tidak diperlukan
36 karena tujuan utamanya adalah validitas tautan, bukan isi konten.

37 Dengan pertimbangan tersebut, aplikasi pemeriksa tautan lebih sesuai menggunakan pendekatan
38 *focused crawling* dengan batasan pada domain yang sama.

1 Tantangan Teknis

2 Subbab 2.4.3 menjelaskan sejumlah tantangan yang umum dihadapi dalam *crawling*. Dalam
3 sistem yang dikembangkan, setiap tantangan tersebut ditinjau kembali dan ditetapkan keputusan
4 penerapannya sebagai berikut:

- 5 • **Fetching:** dalam sistem ini akan diterapkan pengaturan *timeout* agar proses tidak berhenti
6 terlalu lama pada tautan yang tidak merespons. Selain itu, akan ada jeda antar permintaan
7 sebagai bentuk pengendalian agar server tidak terbebani dan tidak memblokir pemeriksaan.
- 8 • **Parsing:** dalam sistem ini akan digunakan Jsoup untuk mengurai HTML. Parser ini dipilih
9 karena mampu menangani dokumen dengan struktur yang tidak sempurna, sehingga tautan
10 tetap dapat diekstrak. Namun, tautan yang dihasilkan secara dinamis melalui JavaScript
11 tidak akan diperiksa karena sistem hanya memproses HTML statis.
- 12 • **Link Extraction dan Canonicalization:** dalam sistem ini semua URL yang ditemukan
13 akan dinormalisasi. Langkah ini dilakukan agar tidak ada duplikasi, misalnya akibat perbedaan
14 huruf besar, tanda garis miring di akhir, atau adanya fragmen. Dengan begitu, satu sumber
15 daya tidak akan dianggap berbeda hanya karena variasi penulisan.
- 16 • **Repository:** dalam sistem ini semua URL yang sudah ditemukan, baik halaman maupun
17 sumber daya lain seperti gambar, skrip, dan stylesheet, akan disimpan di dalam repository.
18 Dengan cara ini, tidak ada tautan yang sama diperiksa ulang dan proses pemeriksaan menjadi
19 lebih efisien.
- 20 • **Spider Trap dan Infinite Loops:** dalam sistem ini tidak akan diterapkan mekanisme
21 khusus untuk mendeteksi pola tautan tak terbatas. Namun, repository sudah cukup untuk
22 mencegah kunjungan berulang pada URL yang sama. Pengecualian hanya berlaku untuk
23 kasus *redirect loop*, yang tetap akan diperiksa agar sistem tidak terjebak mengikuti *redirect*
24 tanpa akhir.
- 25 • **Concurrency:** dalam sistem ini *concurrency* akan diterapkan, tetapi hanya pada tahap
26 pemeriksaan tautan (HEAD atau GET) agar proses lebih cepat. Untuk proses *crawling*
27 halaman dan parsing HTML tetap dilakukan secara berurutan agar hasil ekstraksi lebih
28 terkontrol. Jumlah permintaan paralel akan dibatasi supaya tidak menimbulkan pemblokiran
29 atau respons “too many requests” dari server.

30 Etika *crawling*

31 Subbab 2.4.4 menjelaskan bahwa aktivitas *crawling* tidak hanya berkaitan dengan tantangan teknis,
32 tetapi juga harus memperhatikan etika agar tidak menimbulkan masalah bagi pemilik situs. Salah
33 satu pedoman yang tersedia adalah file **robots.txt**, yang dapat digunakan untuk menentukan
34 bagian situs mana yang boleh dan tidak boleh diakses oleh *crawler*. Pada sistem pemeriksa tautan
35 rusak, aturan ini tidak dijadikan batasan mutlak karena tujuan utama adalah memastikan semua
36 tautan dapat diperiksa. Namun, untuk menjaga sikap yang baik terhadap pemilik situs, keberadaan
37 **robots.txt** tetap dapat dipertimbangkan sebagai acuan tambahan.

38 Aspek lain yang penting adalah identitas **User-Agent**. Setiap permintaan HTTP yang dikirimkan
39 sistem akan dilengkapi dengan informasi ini agar server mengetahui perangkat lunak apa yang sedang
40 melakukan *crawling*. Identitas tersebut sebaiknya mencantumkan nama aplikasi dan informasi
41 kontak, sehingga administrator situs dapat mengenali sumber permintaan dengan jelas.

1 Selain itu, sistem tidak dirancang untuk mengakses area privat, melewati autentikasi, atau
2 mengambil konten yang bersifat berbayar. Aktivitas *crawling* difokuskan pada tautan yang memang
3 dapat diakses secara publik, sehingga tidak melanggar aturan kepemilikan maupun hak akses.
4 Untuk mencegah server terbebani, pengaturan batas waktu dan laju permintaan juga digunakan.
5 Dengan cara ini, sistem tetap menjalankan prinsip etika dasar dalam *crawling* sekaligus mencapai
6 tujuannya dalam memeriksa ketersediaan tautan.

7 **Ekstraksi Tautan**

8 Subbab ?? mencatat berbagai elemen HTML yang memiliki atribut berisi URL. Dalam konteks
9 aplikasi pemeriksa tautan rusak, tidak semua elemen tersebut relevan. Analisis ini menentukan
10 elemen mana yang diekstrak, atribut apa yang digunakan, serta alasan pemilihannya. Elemen-elemen
11 yang dipilih adalah sebagai berikut:

- 12 • <a>: menggunakan atribut **href** sebagai tautan utama antarhalaman. Elemen ini menjadi
13 sumber navigasi paling penting sehingga wajib diperiksa.
- 14 • <area>: menggunakan atribut **href** pada *image map*. Meskipun jarang dipakai, elemen ini
15 tetap berfungsi sebagai tautan dan perlu diperiksa.
- 16 • <link>: menggunakan atribut **href** untuk menghubungkan dokumen dengan stylesheet, ikon,
17 atau sumber daya eksternal lain. Jika rusak, tampilan halaman dapat terganggu.
- 18 • <script>: atribut **src** menunjuk ke berkas JavaScript eksternal. Tautan ini diperiksa karena
19 jika rusak, fungsi interaktif halaman tidak berjalan.
- 20 • : atribut **src** memuat lokasi gambar. Tautan rusak membuat gambar gagal ditampilkan
21 sehingga perlu dicek.
- 22 • <iframe>: atribut **src** digunakan untuk menyematkan halaman lain di dalam halaman utama.
23 Jika rusak, konten yang diembed tidak muncul.
- 24 • <embed>: menggunakan atribut **src** untuk konten eksternal seperti multimedia. Rusak berarti
25 konten tidak dapat dimuat.
- 26 • <object>: atribut **data** menunjuk ke objek eksternal seperti PDF. Karena sering dipakai
27 pada situs institusi, tautan ini harus diperiksa.
- 28 • <source>: atribut **src** digunakan dalam elemen <audio> atau <video> sebagai alternatif
29 sumber media. Jika rusak, pemutaran media gagal.
- 30 • <track>: atribut **src** menyediakan berkas teks untuk subtitle atau caption. Rusak berarti
31 fitur aksesibilitas tidak berfungsi.
- 32 • <audio>: atribut **src** menunjuk ke berkas audio. Jika rusak, konten audio tidak dapat diputar.
- 33 • <video>: atribut **src** menunjuk ke berkas video. Jika rusak, konten video gagal ditampilkan.
- 34 • <input type="image">: atribut **src** menunjuk ke gambar tombol kirim. Jika rusak, tombol
35 tidak muncul di antarmuka.

36 **3.2 Analisis Sistem Serupa**

37 Untuk memperoleh gambaran mengenai pendekatan yang telah digunakan dalam memeriksa tautan
38 rusak pada situs web, penelitian ini meninjau beberapa sistem serupa yang tersedia secara daring.
39 Peninjauan ini bertujuan untuk memahami fitur dan kemampuan dari sistem-sistem tersebut sehingga

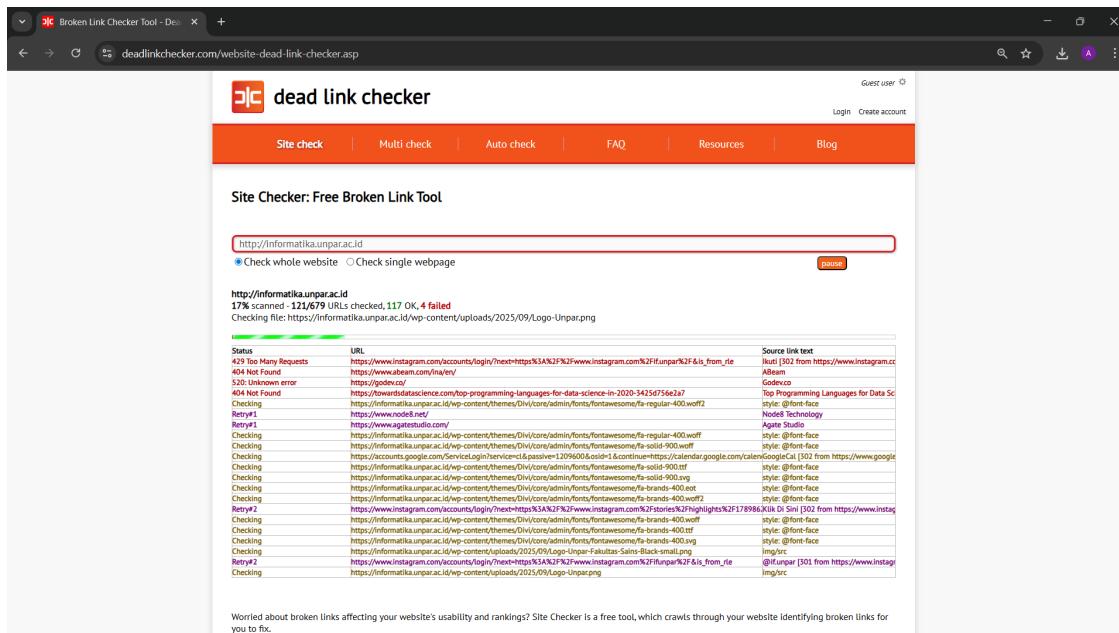
- ¹ dapat menjadi acuan dalam merumuskan kebutuhan fungsional aplikasi yang akan dikembangkan.
 - ² Adapun sistem yang dianalisis adalah Dead Link Checker dan Broken Link Checker, yang masing-masing memiliki karakteristik dan pendekatan berbeda dalam mendeteksi serta melaporkan tautan rusak.

5 3.2.1 Dead Link Checker

- 6 Dead Link Checker¹ adalah sebuah situs web yang dikembangkan oleh DLC Websites untuk
7 mendeteksi tautan rusak pada situs web. Situs ini memiliki tiga layanan utama, yaitu *site check*
8 yang tersedia secara gratis, serta *multi check* dan *auto check* yang tersedia secara berbayar. Layanan
9 *site check* digunakan untuk memeriksa tautan pada satu situs web, *multi check* memungkinkan
10 pemeriksaan pada beberapa situs sekaligus, sedangkan *auto check* menyediakan pemeriksaan berkala
11 dengan laporan hasil yang dikirim melalui email.

12 Tampilan dan Interaksi

- ¹³ Gambar 3.1 menunjukkan tampilan antarmuka Dead Link Checker pada layanan *site check* saat
¹⁴ proses pemeriksaan berlangsung. Alur penggunaan secara umum dimulai dengan memasukkan
¹⁵ alamat situs, memilih mode pemeriksaan, lalu menekan tombol *check* untuk memulai proses.
¹⁶ Selama pemeriksaan berjalan, sistem menampilkan ringkasan progres dan hasilnya diperbarui secara
¹⁷ langsung dalam bentuk tabel.



Gambar 3.1: Antarmuka Dead Link Checker

- 18 Berikut adalah penjelasan komponen antarmuka pengguna pada layanan *site check* :

19 **1. Input URL**

20 Pengguna cukup memasukkan alamat domain atau halaman dengan URL absolut yang ingin

21 diperiksa. Jika skema tidak dituliskan, sistem secara otomatis menambahkan awalan `http://`.

¹ <https://www.deadlinkchecker.com> (Diakses pada 30 Agustus 2025)

1 Hal ini menyederhanakan input, meskipun dapat menimbulkan masalah pada situs yang hanya
2 mendukung skema HTTPS.

3 **2. Mode pemeriksaan**

4 Dead Link Checker menyediakan dua opsi, yakni *Check whole website* untuk menelusuri
5 seluruh halaman dalam domain yang sama dengan URL input, dan *Check single webpage*
6 untuk memeriksa hanya halaman yang diberikan sesuai URL input.

7 **3. Kontrol proses**

8 Pemeriksaan tautan dijalankan dengan menekan tombol *check*. Setelah proses berlangsung,
9 tombol ini berubah menjadi *pause* yang jika ditekan maka akan menghentikan sementara
10 proses pemeriksaan dan menyimpan posisi terakhir. Pada kondisi jeda, tombol *pause* akan
11 digantikan dengan dua tombol, yaitu *resume* untuk melanjutkan pemeriksaan dari titik
12 terakhir, dan *cancel* untuk membatalkan seluruh proses.

13 **4. Ringkasan pemeriksaan**

14 Ringkasan pemeriksaan menampilkan informasi utama terkait progres yang sedang berlangsung.
15 Bagian ini memuat alamat situs yang diperiksa, persentase pemindaian yang telah selesai,
16 jumlah tautan yang sudah diperiksa dibandingkan dengan total tautan yang ditemukan, serta
17 jumlah tautan yang berstatus OK dan yang *failed*. Informasi tambahan juga ditampilkan
18 berupa alamat tautan yang sedang diperiksa pada saat itu. Tepat di bawahnya terdapat
19 *progress bar* berwarna, di mana warna hijau beraksen putih menunjukkan jumlah tautan yang
20 berstatus OK dan sedang di proses, sedangkan warna merah menunjukkan jumlah tautan
21 yang gagal diakses.

22 **5. Tabel hasil**

23 Hasil pemeriksaan ditampilkan dalam bentuk tabel dengan tiga kolom utama yang isinya
24 diperbarui secara langsung (*real time*) selama pemeriksaan berlangsung. Tautan yang valid
25 dihapus otomatis dari tabel setelah berhasil diperiksa, sehingga yang tersisa hanya entri yang
26 bermasalah atau sedang diproses. Berikut adalah penjelasan untuk tiap kolom:

- 27 • **Status:** Pada kolom ini, awalnya semua tautan diberi status *Checking*, jika pada tahap
28 ini terjadi kegagalan koneksi atau tidak ada respons dari server, sistem secara otomatis
29 melakukan percobaan ulang yang ditandai dengan *Retry#1* dan *Retry#2*. Apabila
30 setelah dua kali percobaan ulang tautan tetap tidak dapat diakses, maka status akhir
31 ditetapkan sesuai kondisi terakhir, seperti *Timeout* atau *Host not found*. Sebaliknya,
32 jika server memberikan respons yang jelas, sistem langsung menampilkan kode hasil
33 tanpa melakukan retry, seperti *404 Not Found*, *500 Internal Server Error*, *429 Too*
34 *Many Requests*, atau kode non-standar seperti *999*. Warna latar baris juga digunakan
35 untuk memperjelas status: kuning untuk *Checking*, ungu untuk *Retry*, dan merah untuk
36 tautan yang rusak.
- 37 • **URL:** Kolom ini berisi alamat tautan yang sedang atau sudah diperiksa. Setiap entri
38 dapat diklik untuk membuka alamat tersebut secara langsung di *browser*.
- 39 • **Source link text:** Kolom ini menunjukkan teks jangkar dari tautan atau konteks
40 halaman sumber tempat tautan ditemukan. Bagian ini juga dapat diklik untuk membuka
41 halaman asal tautan ditemukan.

1 Mekanisme dan Ketentuan Teknis

2 Selain fitur yang terlihat pada antarmuka, Dead Link Checker juga memiliki mekanisme dan
3 ketentuan teknis berikut:

4 1. *Crawling* rekursif

5 Dead Link Checker bekerja dengan cara *crawling* halaman web secara rekursif, yaitu dengan
6 mengikuti setiap tautan yang ditemukan untuk kemudian dipindai kembali. Proses dimulai
7 dari URL awal, lalu setiap tautan dalam domain yang sama diperiksa dan jika valid akan
8 diperlakukan sebagai halaman baru untuk dianalisis. Pola ini terus diulangi hingga batas
9 kedalaman tertentu, di mana pemeriksaan penuh (*full scan*) dapat menjangkau hingga sepuluh
10 tingkat halaman.

11 2. Kepatuhan terhadap Robots.txt

12 Dead Link Checker melakukan pemindaian dengan tetap menghormati aturan pada berkas
13 `robots.txt` yang dimiliki oleh situs web target. Pada berkas ini, administrator situs dapat
14 menentukan direktori yang tidak boleh diakses oleh crawler atau memberi jeda antar permintaan
15 untuk mengurangi beban server. Dead Link Checker menggunakan *user-agent* khusus
16 `www.deadlinkchecker.com`, sehingga aturan yang ditulis dengan *user-agent* ini akan dipatuhi.
17 Contoh aturan yang dapat ditambahkan adalah:

```
18 User-agent: www.deadlinkchecker.com
19 Disallow: /shoppingbasket/
20 Crawl-delay: 1
```

21 Instruksi tersebut akan mencegah *crawler* mengakses direktori `/shoppingbasket/` beserta
22 subdirektorinya, serta memaksa jeda minimal satu detik antar permintaan. Dengan mekanisme
23 ini, pemilik situs web memiliki kendali untuk membatasi sejauh mana Dead Link Checker
24 dapat melakukan *crawling* pada situs web mereka.

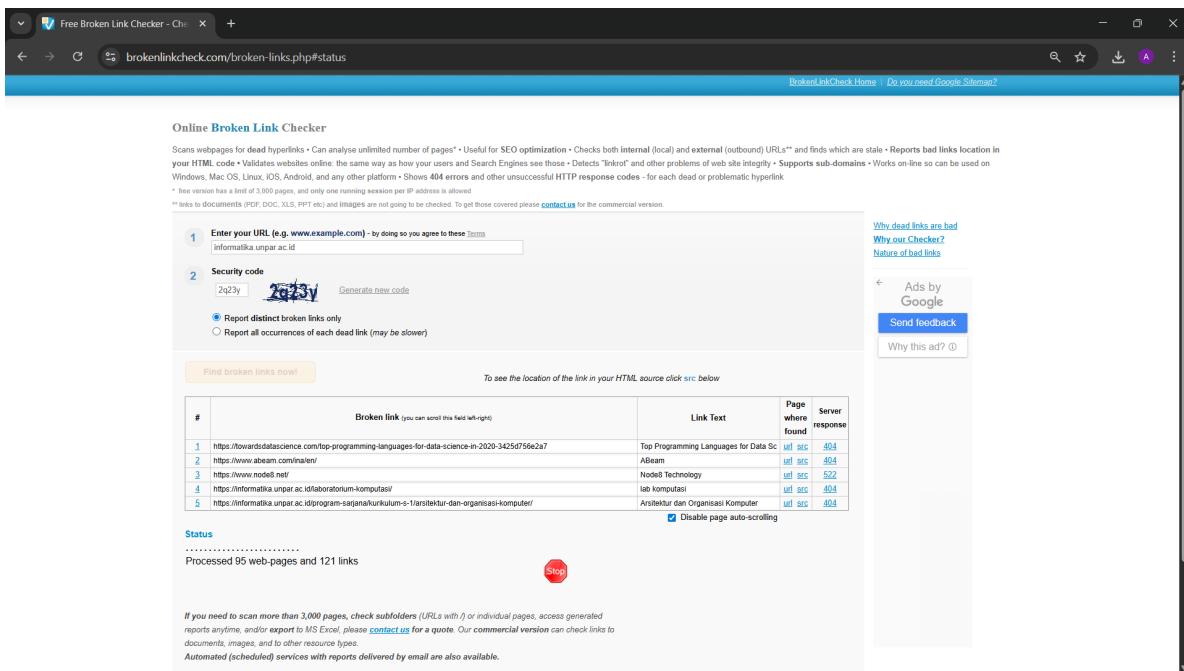
25 3.2.2 Broken Link Checker

26 Broken Link Checker² adalah sebuah situs web yang digunakan untuk mendeteksi tautan rusak,
27 baik tautan internal maupun eksternal pada sebuah situs web.

28 Tampilan dan Interaksi

29 Gambar 3.2 menunjukkan antarmuka Broken Link Checker saat proses pemeriksaan berlangsung.
30 Alur penggunaan secara umum dimulai dengan memasukkan alamat situs pada kolom input, mengisi
31 kode keamanan, memilih mode pemeriksaan, lalu menekan tombol *Find broken links now!* untuk
32 memulai proses. Selama pemeriksaan berjalan, hasil ditampilkan langsung dalam bentuk tabel, dan
33 pengguna dapat menghentikan proses dengan menekan tombol *Stop*.

²<https://www.brokenlinkcheck.com> (Diakses pada 30 Agustus 2025)



Gambar 3.2: Antarmuka Broken Link Checker

1 Berikut adalah komponen utama antarmuka pengguna Broken Link Checker pada layanan
2 pemeriksaan tautan rusak:

3.1. Input URL

4 Komponen ini digunakan untuk memasukkan alamat situs web yang akan diperiksa. Pengguna
5 dapat menuliskan alamat dalam bentuk domain saja, tanpa harus menyertakan skema **http://**
6 atau **https://**. Sistem akan tetap menerima input tersebut dan mengolahnya sebagai URL
7 awal pemeriksaan. Kemudahan ini mempersingkat proses input, meskipun dapat menimbulkan
8 potensi ambiguitas pada situs web yang hanya mendukung skema tertentu, misalnya hanya
9 **https://**.

10.2. Security code

11 Sebelum memulai proses pemeriksaan, pengguna diwajibkan mengisi *security code* berupa
12 captcha. Keharusan ini berfungsi sebagai mekanisme pencegahan terhadap penggunaan
13 otomatis (bot) yang dapat membebani server layanan. Kode keamanan ini bersifat dinamis
14 dan dapat diganti dengan menekan tombol *Generate new code* yang tersedia pada antarmuka.

15.3. Mode pemeriksaan

16 Tersedia dua pilihan mode pemeriksaan yang dapat dipilih melalui tombol radio, yaitu *Report*
17 *distinct broken links only*, dan *Report all occurrences of each dead link*. Mode *Report distinct*
18 *broken links only* menampilkan setiap tautan rusak hanya sekali, tanpa memperhatikan
19 berapa banyak halaman yang memuat tautan tersebut. Pendekatan ini membuat proses
20 lebih cepat, tetapi informasi lokasi kemunculan tautan hanya terbatas pada satu halaman.
21 Sebaliknya, mode *Report all occurrences of each dead link (may be slower)* menampilkan
22 seluruh kemunculan tautan rusak beserta halaman sumbernya. Dengan demikian, jika sebuah
23 tautan rusak terdapat pada beberapa halaman, maka akan muncul beberapa entri dengan
24 sumber berbeda pada tabel hasil. Mode ini memberikan informasi yang lebih lengkap, namun
25 membutuhkan waktu pemrosesan yang lebih lama.

1 4. Tombol kontrol

2 Proses pemeriksaan dimulai dengan menekan tombol *Find broken links now!*. Setelah pemeriksaan berjalan, tombol *Stop* berwarna merah akan muncul di bawah tabel hasil. Tombol ini memungkinkan pengguna menghentikan proses pemeriksaan kapan saja sesuai kebutuhan. Selain itu, disediakan pula opsi *Disable page auto-scrolling* yang dapat diaktifkan untuk nonaktifkan penguliran otomatis pada tabel hasil, sehingga tampilan tetap stabil meskipun data baru terus ditambahkan secara real-time.

8 5. Tabel hasil

9 Hasil pemeriksaan ditampilkan dalam bentuk tabel yang diperbarui secara langsung. Tabel ini hanya menampilkan tautan yang telah dipastikan bermasalah, sehingga laporan lebih fokus dan tidak bercampur dengan tautan valid atau sedang dalam proses pemeriksaan. Terdapat lima kolom utama pada tabel hasil, yaitu:

- 13 • #: kolom nomor urut hasil. Setiap nomor dapat diklik untuk membuka tautan rusak yang bersangkutan secara langsung.
- 14 • **Broken link:** berisi alamat tautan yang teridentifikasi rusak. Tautan ini ditampilkan dalam bentuk teks biasa tanpa fungsi klik.
- 15 • **Link text:** menampilkan teks jangkar (*anchor text*) dari tautan yang ditemukan, atau keterangan terkait jika tersedia.
- 16 • **Page where found:** berisi dua tautan, yaitu *url* yang mengarah ke halaman web tempat tautan rusak ditemukan, serta *src* yang menunjuk ke dokumen HTML sumber untuk memperlihatkan lokasi tautan yang bermasalah.
- 17 • **Server response:** menampilkan status akhir dari hasil pemeriksaan tautan, baik berupa kode status HTTP seperti 404 dan 500, maupun pesan kesalahan non-HTTP seperti **bad host** atau **timeout**.

25 6. Ringkasan pemeriksaan

26 Pada bagian bawah tabel, sistem menampilkan ringkasan proses pemeriksaan yang sedang berlangsung. Bagian ini memuat status terkini, yang ditandai dengan animasi titik berjalan selama proses aktif. Jika pemeriksaan dihentikan oleh pengguna melalui tombol *Stop*, status akan berubah menjadi **DONE : process was terminated by user**. Sebaliknya, jika pemeriksaan selesai secara normal, status ditampilkan sebagai **COMPLETED!**. Selain menampilkan status, ringkasan ini juga memperlihatkan jumlah total halaman dan tautan yang telah diperiksa, serta jumlah tautan rusak yang ditemukan.

33 33. Mekanisme dan Ketentuan Teknis

34 Selain fitur yang tampak pada antarmuka, Broken Link Checker juga memiliki sejumlah mekanisme teknis yang menentukan cara kerja dan batasan layanan. Mekanisme ini penting untuk dipahami agar pengguna dapat menafsirkan hasil pemeriksaan dengan tepat dan menyesuaikan penggunaan sesuai kebutuhan. Mekanisme dan ketentuan teknis tersebut dapat dirangkum sebagai berikut:

38 1. Cakupan pemeriksaan

39 Broken Link Checker melakukan pemeriksaan terhadap tautan internal maupun eksternal dari sebuah situs web. Tautan internal adalah tautan yang berada dalam domain yang sama dengan URL awal, sedangkan tautan eksternal adalah tautan yang mengarah keluar ke domain lain.

1 Hanya tautan yang bermasalah yang ditampilkan dalam laporan, sedangkan tautan valid tidak
2 ditampilkan. Pendekatan ini membuat hasil pemeriksaan lebih bersih dan mudah dianalisis
3 karena pengguna tidak perlu memilah-milah tautan yang sebenarnya masih berfungsi.

4 **2. Batasan versi gratis**

5 Versi gratis dari layanan ini memiliki beberapa pembatasan. Pemeriksaan hanya dapat
6 dilakukan hingga maksimum 3.000 halaman dalam satu sesi, dengan ketentuan satu sesi
7 diperbolehkan untuk setiap alamat IP pada satu waktu. Selain itu, pemeriksaan pada versi
8 gratis tidak mencakup tautan yang menuju ke dokumen seperti PDF, DOC, XLS, PPT,
9 maupun tautan ke file gambar. Batasan ini diberlakukan untuk mengurangi beban layanan
10 sekaligus mendorong pengguna yang membutuhkan cakupan lebih luas untuk beralih ke versi
11 komersial.

12 **3. Fitur versi komersial**

13 Untuk kebutuhan yang lebih kompleks, Broken Link Checker menyediakan layanan berbayar
14 dengan fitur tambahan. Pada versi ini, pemeriksaan dapat dilakukan pada jumlah halaman
15 yang jauh lebih besar tanpa batasan ketat seperti pada versi gratis. Selain itu, layanan berbayar
16 mendukung pemindaian sub-folder dan sub-domain, pemeriksaan tautan yang mengarah ke
17 dokumen maupun gambar, serta ekspor hasil pemeriksaan ke format CSV atau Excel untuk
18 memudahkan analisis lanjutan. Fitur lain yang ditawarkan mencakup pembuatan laporan
19 otomatis yang dikirim melalui email secara terjadwal (harian, mingguan, atau bulanan), opsi
20 konfigurasi kecepatan pemeriksaan sesuai kebutuhan situs target, serta analisis tambahan
21 seperti pemeriksaan feed RSS atau ATOM.

22 **3.3 Analisis Kebutuhan**

23 Analisis masalah pada Subbab 3.1, serta analisis sistem serupa pada Subbab 3.2, menghasilkan
24 gambaran mengenai fitur dan karakteristik yang harus dimiliki oleh sistem. Dari sini disusun
25 kebutuhan fungsional yang menjelaskan layanan apa saja yang harus disediakan oleh sistem,
26 serta kebutuhan non-fungsional yang berhubungan dengan kualitas, batasan teknis, dan kriteria
27 pendukung lainnya.

28 Gambaran umum dari kebutuhan sistem dapat divisualisasikan dalam bentuk diagram *use case*
29 sebagaimana ditunjukkan pada Gambar 3.3. Diagram tersebut memperlihatkan aktor utama yaitu
30 pengguna yang berinteraksi langsung dengan perangkat lunak yang dikembangkan. Terdapat dua
31 layanan inti yang dapat dilakukan pengguna, yaitu melakukan pemeriksaan tautan rusak pada
32 sebuah situs web dan mengekspor hasil pemeriksaan ke dalam berkas eksternal.

33 **1. Memeriksa Tautan Rusak**

34 Fitur ini digunakan untuk melakukan pemeriksaan terhadap seluruh tautan yang terdapat
35 dalam sebuah situs web untuk mendeteksi tautan rusak (broken links).

- 36 • Nama: Memeriksa Tautan Rusak
37 • Aktor: Pengguna
38 • Deskripsi: Memeriksa tautan rusak yang ada pada sebuah situs web.
39 • Kondisi awal: Aplikasi telah dijalankan dan pengguna berada pada jendela utama.
40 • Kondisi akhir: Daftar tautan rusak ditampilkan di tabel hasil pemeriksaan.
41 • Skenario utama: Ditampilkan dalam tabel 3.1

Tabel 3.1: Skenario Memeriksa Tautan Rusak

No	Aksi Aktor	Reaksi Sistem
1	Pengguna memasukkan URL ke dalam kolom masukan dan menekan tombol <i>Check</i> .	Sistem melakukan proses <i>web crawling</i> , mengekstrak seluruh tautan yang ditemukan dan mengecek kode status HTTP dari setiap tautan.
2		Sistem menampilkan daftar tautan rusak yang ditemukan dan ringkasan dari proses pengecekan.

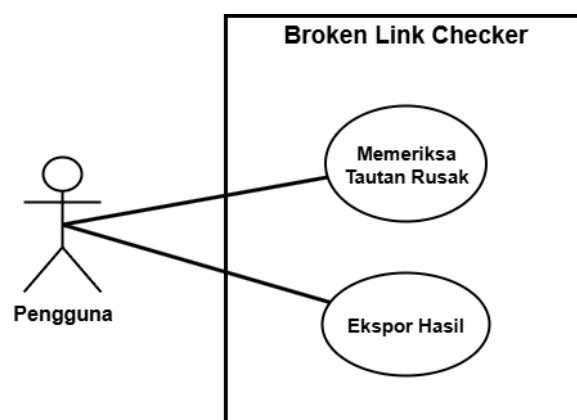
2. Ekspor Hasil

Fitur ini digunakan untuk mengekspor hasil pemeriksaan tautan rusak ke dalam berkas eksternal untuk keperluan dokumentasi atau analisis lebih lanjut.

- Nama: Ekspor Hasil
- Aktor: Pengguna
- Deskripsi: Ekspor hasil pemeriksaan tautan rusak ke berkas eksternal dalam format Excel.
- Kondisi awal: Pemeriksaan tautan telah selesai dan hasilnya telah ditampilkan di tabel.
- Kondisi akhir: Berkas hasil ekspor tersimpan di perangkat pengguna.
- Skenario utama: Ditampilkan dalam tabel 3.2

Tabel 3.2: Skenario Ekspor Hasil

No	Aksi Aktor	Reaksi Sistem
1	Pengguna menekan tombol <i>Export</i> .	Sistem membuka dialog penyimpanan berkas.
2	Pengguna menentukan nama dan lokasi penyimpanan berkas.	Sistem memproses data hasil pemeriksaan menjadi format .xlsx.
3		Sistem menyimpan berkas hasil ekspor di lokasi yang telah dipilih pengguna.

Gambar 3.3: Diagram *use case* aplikasi

¹ **3.3.1 Kebutuhan Fungsional**

² Kebutuhan fungsional berikut menggambarkan layanan inti yang harus disediakan oleh aplikasi
³ desktop pemeriksa tautan rusak pada situs web:

⁴ **1. Input URL**

⁵ Sistem harus dapat menerima masukan berupa URL dari pengguna sebagai titik awal pemeriksaan.
⁶

⁷ **2. Tampilan hasil secara *streaming***

⁸ Sistem menampilkan daftar tautan yang ditemukan dari halaman web secara langsung (*streaming*) selama proses pemeriksaan berlangsung.
⁹

¹⁰ **3. Penghentian proses**

¹¹ Sistem menyediakan mekanisme bagi pengguna untuk menghentikan proses pemeriksaan kapan saja.
¹²

¹³ **4. Mekanisme *retry***

¹⁴ Sistem melakukan percobaan ulang terhadap tautan yang gagal diperiksa akibat kesalahan sementara, misalnya *timeout* atau gangguan jaringan.
¹⁵

¹⁶ **5. Ringkasan hasil**

¹⁷ sistem menampilkan ringkasan berisi jumlah tautan yang diperiksa, jumlah tautan rusak, jumlah tautan yang terblokir, dan status proses pemeriksaan.
¹⁸

¹⁹ **6. Sumber tautan rusak**

²⁰ Sistem menampilkan lokasi atau halaman tempat tautan rusak ditemukan, sehingga pengguna dapat menelusuri asal masalah.
²¹

²² **7. Ekspor hasil**

²³ Sistem dapat mengekspor hasil pemeriksaan ke dalam berkas Excel agar mudah disimpan, dibagikan, atau dianalisis lebih lanjut.
²⁴

²⁵ **3.3.2 Kebutuhan Non-Fungsional**

²⁶ Selain fungsi inti, sistem juga harus memenuhi sejumlah kebutuhan non-fungsional agar pemeriksaan tautan dapat berjalan stabil, efisien, dan mudah digunakan. Kebutuhan non-fungsional yang ditetapkan adalah sebagai berikut:

²⁹ **1. Antarmuka pengguna**

³⁰ Sistem disajikan dalam bentuk aplikasi desktop berbasis JavaFX dengan tampilan yang sederhana, tabel yang mudah dibaca, serta kontrol proses yang jelas.
³¹

³² **2. Kinerja pemeriksaan**

³³ Sistem mampu menangani jumlah tautan yang besar dengan tetap menjaga waktu tanggap yang wajar. Hal ini dicapai dengan penerapan batas waktu (*timeout*) pada setiap permintaan.
³⁴

³⁵ **3. Pengendalian laju**

³⁶ Sistem membatasi jumlah permintaan dalam satuan waktu (*rate limiting*) untuk mencegah server tujuan terbebani dan mengurangi risiko pemblokiran.
³⁷

³⁸ **4. Eksekusi paralel**

³⁹ Sistem mendukung eksekusi permintaan secara paralel (*concurrency*) khusus pada tahap pemeriksaan tautan agar proses lebih cepat. Jumlah permintaan simultan dibatasi agar tetap stabil dan tidak menimbulkan masalah 429 Too Many Requests atau pemblokiran.
⁴⁰
⁴¹

1 **5. Ketahanan terhadap HTML tidak valid**

2 Sistem mampu mengurai dokumen HTML yang tidak sesuai standar dengan bantuan Jsoup,
3 sehingga tautan tetap dapat diekstrak meskipun struktur dokumen bermasalah.

4 **6. Konsistensi identifikasi URL**

5 Setiap URL dinormalisasi dan dicatat dalam repository untuk memastikan satu sumber daya
6 hanya diperiksa sekali, sehingga hasil pemeriksaan konsisten dan tidak ada duplikasi.

7 **7. Pelabelan hasil yang jelas**

8 Setiap hasil pemeriksaan ditampilkan dengan kode status HTTP beserta keterangan resminya,
9 serta pesan khusus untuk kesalahan non-HTTP seperti *timeout*, *bad host*, atau kesalahan
10 koneksi.

11 **8. Etika crawling**

12 Setiap permintaan menyertakan identitas **User-Agent**. Sistem juga menyediakan pengaturan
13 jeda permintaan untuk menjaga agar aktivitas pemeriksaan tidak membebani server.

14

3.4 Analisis Teknologi yang Digunakan

15

3.4.1 JavaFX

16 Pemilihan JavaFX sebagai pustaka antarmuka pengguna didasarkan pada kebutuhan sistem yang
17 harus disajikan dalam bentuk aplikasi desktop dengan tampilan sederhana, tabel yang mudah dibaca,
18 serta kontrol proses yang jelas (lihat Subbabab [3.3.2](#)). Sebagaimana dijelaskan pada Subbab [2.5](#),
19 JavaFX menawarkan paradigma *scene graph* yang memungkinkan penyusunan antarmuka secara
20 hierarkis dan konsisten. Paradigma ini memudahkan pengelolaan komponen visual yang diperlukan
21 untuk menampilkan hasil pemeriksaan tautan secara *streaming* (lihat Subbabab [3.3.1](#)).

22 Dalam implementasi, JavaFX akan digunakan melalui beberapa komponen utama berikut:

- **Stage, Scene, dan Node.** Struktur dasar aplikasi akan mengikuti siklus hidup JavaFX dengan menurunkan kelas dari **Application**. Objek **Stage** akan berperan sebagai jendela utama, sedangkan **Scene** digunakan untuk menampung keseluruhan antarmuka. Komponen UI seperti **Button**, **Label**, dan **TextField** direpresentasikan sebagai turunan **Node** yang diorganisasi dalam kontainer tata letak, misalnya **VBox** atau **BorderPane**.
- **FXML dan Controller.** Untuk memisahkan logika aplikasi dari tampilan, struktur antarmuka didefinisikan secara deklaratif dalam berkas FXML. Atribut **fx:id** akan dipakai agar komponen UI dapat diakses dari kelas controller, sedangkan event handler seperti **onAction** digunakan untuk menangani interaksi pengguna, misalnya saat memulai atau menghentikan proses pemeriksaan tautan.
- **Komponen Tabel.** Hasil pemeriksaan tautan akan ditampilkan dalam bentuk tabel menggunakan **TableView**. Komponen ini mendukung penyajian data terstruktur dalam baris dan kolom, sehingga cocok untuk menampilkan daftar halaman yang diperiksa maupun daftar tautan rusak. Setiap kolom akan diikat dengan **Property** pada model data agar perubahan nilai dapat langsung tercermin pada tampilan.
- **Property dan Binding.** Untuk mendukung pembaruan data secara langsung, mekanisme **StringProperty**, **BooleanProperty**, dan **IntegerProperty** akan digunakan. Binding dua arah dimanfaatkan agar nilai pada komponen input dan model selalu konsisten, sedangkan

binding satu arah memastikan perubahan status pemeriksaan langsung diperlihatkan pada label atau tabel.

- **Pengendalian Thread.** Karena proses pemeriksaan tautan berjalan secara paralel, pembaruan antarmuka pengguna harus dijalankan melalui `Platform.runLater()`. Hal ini menjamin sinkronisasi antara *thread* pemeriksaan dengan *thread* JavaFX, sehingga tampilan dapat diperbarui secara aman tanpa menimbulkan error `Not on FX application thread`.

3.4.2 Jsoup

Pemilihan Jsoup sebagai pustaka utama untuk pengambilan dan pemrosesan dokumen HTML didasarkan pada kebutuhan sistem untuk mengekstraksi tautan dari halaman web dengan cara yang andal, termasuk dari dokumen yang tidak valid atau tidak sepenuhnya sesuai standar. Kebutuhan ini secara eksplisit dinyatakan pada aspek ketahanan terhadap HTML tidak valid (lihat Subbab 3.3.2). Sebagaimana dijelaskan pada Subbab 2.6, Jsoup menyediakan parser yang toleran kesalahan dan mendukung spesifikasi HTML5, sehingga struktur DOM tetap dapat dibentuk meskipun dokumen bermasalah.

Dari sisi fungsional, penggunaan Jsoup relevan dengan kebutuhan untuk:

- menerima masukan URL dan mengambil halaman web terkait (lihat Subbab 3.3.1).
- mengekstraksi seluruh tautan yang terdapat di dalam elemen HTML, khususnya elemen `<a>` dengan atribut `href`.
- menampilkan hasil pemeriksaan secara *streaming*, sehingga setiap tautan yang diperoleh dari hasil parsing dapat segera diperiksa dan ditampilkan ke antarmuka pengguna.

Dalam implementasi, beberapa kelas dan method dari Jsoup akan digunakan secara langsung, yaitu:

- **Jsoup**

Kelas ini dipakai sebagai titik masuk untuk membuat koneksi HTTP ke sebuah halaman melalui `connect(String url)`. Method ini dilanjutkan dengan konfigurasi `userAgent(String ua)` untuk memenuhi etika *crawling* dan `timeout(int millis)` untuk memenuhi kebutuhan non-fungsional terkait batas waktu respon. Eksekusi permintaan dilakukan dengan `get()` atau `execute()`, menghasilkan `Document` atau `Connection.Response`.

- **Connection dan Connection.Response**

`Connection` digunakan untuk mengatur parameter koneksi, sementara `Connection.Response` diperlukan untuk membaca kode status HTTP dengan `statusCode()` serta metadata lain seperti `headers()`. Informasi ini mendukung kebutuhan untuk melabeli hasil pemeriksaan dengan status yang jelas (lihat Subbab 3.3.2).

- **Document**

Objek ini merepresentasikan halaman web yang berhasil diambil. Method `select(String cssQuery)` akan dipakai untuk menemukan seluruh elemen `<a>` dengan atribut `href`. Dari sini dihasilkan objek `Elements`.

- **Elements dan Element**

Koleksi `Elements` berisi banyak `Element` yang masing-masing mewakili sebuah tautan. Method penting yang digunakan adalah `attr("href")` untuk mengambil nilai atribut asli, `text()` untuk isi teks tautan, serta `absUrl("href")` untuk memperoleh URL absolut berdasarkan

1 `baseUri()` dokumen. Normalisasi URL ini mendukung konsistensi identifikasi sumber daya
2 (lihat Subsubbab 3.3.2).

3 **3.4.3 Java HTTP Client API**

4 Selain Jsoup yang digunakan untuk mengambil dan memproses halaman situs web, sistem ini juga
5 membutuhkan mekanisme pemeriksaan terhadap tautan lain, baik tautan eksternal maupun tautan
6 non-halaman (misalnya berkas gambar, skrip, atau sumber daya lain). Untuk kebutuhan ini dipilih
7 Java HTTP Client API, sebagaimana dijelaskan pada Subbab 2.7. API ini merupakan bagian resmi
8 dari Java sejak versi 11, sehingga tidak memerlukan pustaka tambahan dan terintegrasi langsung
9 dengan ekosistem bahasa Java.

10 Pemanfaatan Java HTTP Client API sejalan dengan beberapa kebutuhan non-fungsional,
11 antara lain penerapan batas waktu (*timeout*) agar pemeriksaan tidak menggantung terlalu lama,
12 pengendalian laju permintaan melalui konfigurasi `HttpClient`, serta dukungan eksekusi paralel yang
13 memanfaatkan sifat *thread-safe* dari objek `HttpClient`. Dari sisi fungsional, API ini memungkinkan
14 sistem untuk memperoleh status kode HTTP dari setiap tautan, yang kemudian ditampilkan dengan
15 label yang jelas pada antarmuka (lihat Subsubbab 3.3.1 dan Subsubbab 3.3.2).

16 Beberapa kelas dan method utama dari API ini yang digunakan dalam implementasi adalah
17 sebagai berikut:

18 • **HttpClient**

19 Kelas ini dipakai sebagai titik masuk untuk membangun klien HTTP. Objek dibuat melalui
20 `HttpClient.newBuilder()`, dengan konfigurasi penting seperti `version()` untuk memilih
21 protokol HTTP/1.1 atau HTTP/2, `connectTimeout(Duration)` untuk menetapkan batas
22 waktu koneksi, dan `followRedirects()` untuk menentukan kebijakan pengalihan. Objek yang
23 dihasilkan bersifat *immutable* dan *thread-safe*, sehingga dapat digunakan secara bersamaan
24 oleh banyak *thread* pemeriksaan.

25 • **HttpRequest**

26 Kelas ini digunakan untuk menyusun permintaan HTTP yang akan dikirim. Metode `uri(URI)`
27 menentukan alamat tujuan, sementara `header()` dipakai untuk menyertakan informasi tambahan
28 seperti `User-Agent`. Jenis operasi HTTP ditentukan dengan `method()` atau `GET()`,
29 sedangkan `timeout(Duration)` memastikan setiap permintaan memiliki batas waktu pemrosesan.

30 • **HttpResponse**

31 Objek ini merepresentasikan hasil tanggapan dari server. Informasi utama yang digunakan
32 adalah kode status HTTP melalui `statusCode()`, `header` melalui `headers()`, serta isi respon
33 melalui `body()`. Data ini akan dicatat dan ditampilkan pada antarmuka, termasuk untuk
34 mengidentifikasi tautan rusak (misalnya dengan status 404) atau tautan yang terblokir.

35 • **HttpHeaders**

36 Digunakan untuk membaca seluruh `header` dari respon, misalnya `Content-Type` atau `Set-Cookie`.
37 Informasi ini relevan untuk analisis tambahan, namun yang paling penting adalah mendukung
38 pelabelan hasil pemeriksaan agar pengguna dapat memahami konteks status setiap tautan.

1

BAB 4

2

PERANCANGAN

3 Bab ini membahas perancangan aplikasi desktop pemeriksa tautan rusak pada situs web berdasarkan
4 hasil analisis yang telah dilakukan pada Bab 3. Perancangan mencakup rancangan kelas, perancangan
5 alur dan perancangan antarmuka. Dengan adanya perancangan ini, implementasi dapat dilakukan
6 secara lebih terarah dan sesuai dengan kebutuhan yang telah ditentukan.

7 **4.1 Perancangan Kelas**

8 Struktur kelas pada aplikasi *Broken Link Checker* dibagi ke dalam tiga *package* utama, yaitu *Web*
9 *Crawling*, *URL Fetcher*, dan *GUI Manager*. *Package Web Crawling* berisi kelas-kelas inti yang
10 mengatur proses *crawling*, pengelolaan *frontier*, serta representasi tautan. *Package URL Fetcher*
11 menyediakan mekanisme untuk mengambil konten dari situs web menggunakan pustaka eksternal
12 seperti Jsoup dan HttpClient. Sementara itu, *package GUI Manager* menangani pengelolaan
13 antarmuka pengguna dengan JavaFX, termasuk kontrol proses pemeriksaan dan penyajian hasil.
14 Diagram kelas untuk aplikasi ini dapat dilihat pada Gambar 4.1.

15 Penjelasan setiap kelas pada diagram adalah sebagai berikut:

16 **1. Kelas Link**

17 Kelas *Link* merupakan kelas dasar yang merepresentasikan sebuah tautan dalam sistem. Kelas
18 ini menjadi induk bagi *WebpageLink* dan *BrokenLink*, sehingga relasinya berupa pewarisan.

- 19 • **url** : Atribut ini bertipe data *String* dan digunakan untuk menyimpan alamat dari
20 tautan yang diperiksa.
- 21 • **statusCode** : Atribut ini bertipe data *int* dan berfungsi untuk menyimpan kode status
22 HTTP hasil pemeriksaan tautan.
- 23 • **accessTime** : Atribut ini bertipe data *Instant* dan merekam waktu terakhir kali tautan
24 tersebut diakses.

25 **2. Kelas WebpageLink**

26 Kelas *WebpageLink* merepresentasikan tautan yang termasuk dalam kategori halaman web
27 dengan *host* yang sama seperti URL awal. Kelas ini mewarisi *Link* dan memiliki relasi
28 *many-to-many* dengan *BrokenLink*.

- 29 • **brokenLinks** : Atribut ini bertipe data *Map<BrokenLink, String>* dan digunakan untuk
30 menyimpan daftar tautan rusak yang ditemukan di halaman beserta informasi anchor
31 text-nya.

32 **3. Kelas BrokenLink**

33 Kelas *BrokenLink* digunakan untuk menyimpan data tautan yang gagal diakses atau meng-

1 hasilkan status *error*. Kelas ini mewarisi `Link` dan memiliki relasi *many-to-many* dengan
2 `WebpageLink`.

- 3 • `webpageLinks` : Atribut ini bertipe data `Map<WebpageLink, String>` dan digunakan
4 untuk mencatat daftar halaman tempat tautan rusak tersebut ditemukan beserta lokasi
5 anchor text-nya.

6 **4. Kelas Crawler**

7 Kelas `Crawler` merupakan inti dari proses *web crawling* dan bertanggung jawab mengelola
8 antrean URL, melakukan pengambilan halaman, serta memeriksa status tautan. Kelas ini
9 berhubungan langsung dengan `Frontier`, `WebpageLink`, dan `BrokenLink`, serta memiliki
10 dependensi terhadap `Jsoup` dan `HttpClient`.

- 11 • `rootHost` : Atribut ini bertipe data `String` dan digunakan untuk menyimpan host dari
12 URL awal (seed URL).
- 13 • `frontier` : Atribut ini bertipe data `Frontier` dan berfungsi sebagai antrean URL yang
14 akan diproses.
- 15 • `repository` : Atribut ini bertipe data `Set<String>` dan digunakan untuk menyimpan
16 daftar URL yang sudah pernah dikunjungi agar tidak diperiksa dua kali.
- 17 • `USER_AGENT` : Atribut statis ini bertipe data `String` dan digunakan untuk mendefinisikan
18 identitas permintaan HTTP yang dikirim oleh aplikasi.
- 19 • `TIMEOUT` : Atribut statis ini bertipe data `int` dan menentukan batas waktu maksimum
20 koneksi HTTP.
- 21 • `HTTP_CLIENT` : Atribut statis ini bertipe data `HttpClient` dan digunakan sebagai klien
22 bawaan Java untuk mengirim permintaan HTTP.
- 23 • `Crawler` : Metode ini merupakan konstruktor yang menerima parameter `seedUrl` bertipe
24 data `String` dan digunakan untuk menginisialisasi proses *crawling*.
- 25 • `startCrawling` : Metode ini menerima parameter `streamBrokenLink` bertipe `Consumer<BrokenLink>` dan tidak mengembalikan nilai. Metode ini berfungsi untuk memulai
26 proses *crawling* dan mengirim hasilnya secara *streaming*.
- 27 • `extractLinks` : Metode ini menerima parameter `doc` bertipe `Document` dan mengem-
28 balikan daftar objek `BrokenLink`. Metode ini digunakan untuk mengekstrak seluruh
29 tautan dari halaman HTML yang diambil.
- 30 • `normalizeUrl` : Metode ini menerima parameter `url` bertipe `String` dan mengembalikan
31 hasil normalisasi dalam bentuk `String`. Metode ini digunakan untuk memastikan URL
32 dalam format yang konsisten.
- 33 • `fetchUrl` : Metode ini menerima parameter `url` bertipe `String` dan mengembalikan
34 nilai `int` yang merupakan kode status HTTP hasil pemeriksaan. Metode ini berfungsi
35 untuk melakukan pemeriksaan tautan eksternal atau tautan umum menggunakan Java
36 `HttpClient`.
- 37 • `isPotentialWebpage` : Metode ini menerima parameter `url` bertipe `String` dan meng-
38 embalikan nilai `boolean`. Metode ini digunakan untuk menentukan apakah suatu URL
39 berpotensi merupakan halaman web.

40 **5. Kelas Frontier**

41 Kelas `Frontier` berfungsi sebagai struktur data antrean (*queue*) yang menyimpan URL yang

akan diproses berikutnya. Kelas ini berelasi erat dengan **Crawler** yang mengelola antrean selama proses *crawling* berlangsung.

- **urls** : Atribut ini bertipe data *Queue<String>* dan digunakan untuk menampung daftar URL yang menunggu diproses.
- **enqueue** : Metode ini menerima parameter **url** bertipe *String* dan berfungsi untuk menambahkan URL baru ke dalam antrean.
- **dequeue** : Metode ini tidak menerima parameter dan mengembalikan nilai *String* berupa URL paling depan dari antrean.
- **isEmpty** : Metode ini tidak menerima parameter dan mengembalikan nilai *boolean* untuk menunjukkan apakah antrean kosong atau tidak.

6. Kelas Jsoup dan HttpClient

Kedua kelas ini merupakan representasi dari pustaka eksternal yang digunakan untuk melakukan pengambilan data web dan digunakan sebagai dependensi oleh kelas **Crawler**.

- **Jsoup** : Kelas ini digunakan untuk mengakses dan mengurai dokumen HTML dari halaman same-host dengan memanfaatkan parser HTML toleran kesalahan.
- **HttpClient** : Kelas ini digunakan untuk melakukan permintaan **HEAD** atau **GET** terhadap tautan umum atau eksternal untuk memeriksa status HTTP-nya.

7. Kelas Controller

Kelas **Controller** berfungsi sebagai penghubung antara logika aplikasi dengan antarmuka pengguna yang dibangun menggunakan JavaFX. Kelas ini berinteraksi dengan **Crawler** untuk menginisiasi dan menerima hasil proses pemeriksaan tautan.

- **seedUrlField** : Atribut ini bertipe *TextField* dan digunakan untuk menampung input URL dari pengguna.
- **progressLabel** : Atribut ini bertipe *Label* dan menampilkan status kemajuan proses pemeriksaan.
- **totalLinksLabel** : Atribut ini bertipe *Label* dan menampilkan jumlah total tautan yang diperiksa.
- **webpagesLabel** : Atribut ini bertipe *Label* dan menampilkan jumlah halaman yang berhasil di-crawl.
- **brokenLinksLabel** : Atribut ini bertipe *Label* dan menampilkan jumlah tautan rusak yang ditemukan.
- **resultsTable** : Atribut ini bertipe *TableView<BrokenLink>* dan menampilkan hasil daftar tautan rusak.
- **statusColumn** : Atribut ini bertipe *TableColumn<BrokenLink, String>* dan digunakan untuk menampilkan kolom status pada tabel hasil.
- **urlColumn** : Atribut ini bertipe *TableColumn<BrokenLink, String>* dan digunakan untuk menampilkan kolom URL tautan pada tabel hasil.
- **initialize** : Metode ini tidak menerima parameter dan berfungsi untuk melakukan inisialisasi awal saat antarmuka dimuat.
- **onStartClick** : Metode ini tidak menerima parameter dan digunakan untuk menangani event ketika pengguna menekan tombol *Start*, kemudian memulai proses pemeriksaan tautan.

- 1 • `onStopClick` : Metode ini tidak menerima parameter dan digunakan untuk menghentikan
2 proses pemeriksaan yang sedang berlangsung.
- 3 • `onExportClick` : Metode ini tidak menerima parameter dan digunakan untuk mengekspor
4 hasil pemeriksaan ke dalam berkas Excel.

5 8. Kelas Application

6 Kelas `Application` merupakan titik masuk (*entry point*) dari aplikasi JavaFX. Kelas ini
7 berhubungan dengan `Controller` untuk memuat dan menginisialisasi GUI.

- 8 • `main` : Metode ini tidak menerima parameter dan digunakan untuk menjalankan aplikasi.
- 9 • `start` : Metode ini menerima parameter `stage` bertipe `Stage` dan berfungsi untuk
10 memuat serta menampilkan antarmuka pengguna utama.

11 9. Kelas HttpStatus

12 Kelas `HttpStatus` berfungsi sebagai kelas utilitas yang memetakan kode status HTTP ke
13 *reason phrase*. Kelas ini digunakan oleh `Controller` untuk menampilkan hasil status HTTP
14 pada GUI.

- 15 • `STATUS_MAP` : Atribut ini bertipe `Map<Integer, String>` dan digunakan untuk menyimpan
16 pasangan antara kode status HTTP dengan keterangan resminya.
- 17 • `getStatusCode` : Metode ini menerima parameter `statusCode` bertipe `int` dan mengembalikan
18 nilai `String` berupa keterangan lengkap dari kode status tersebut.

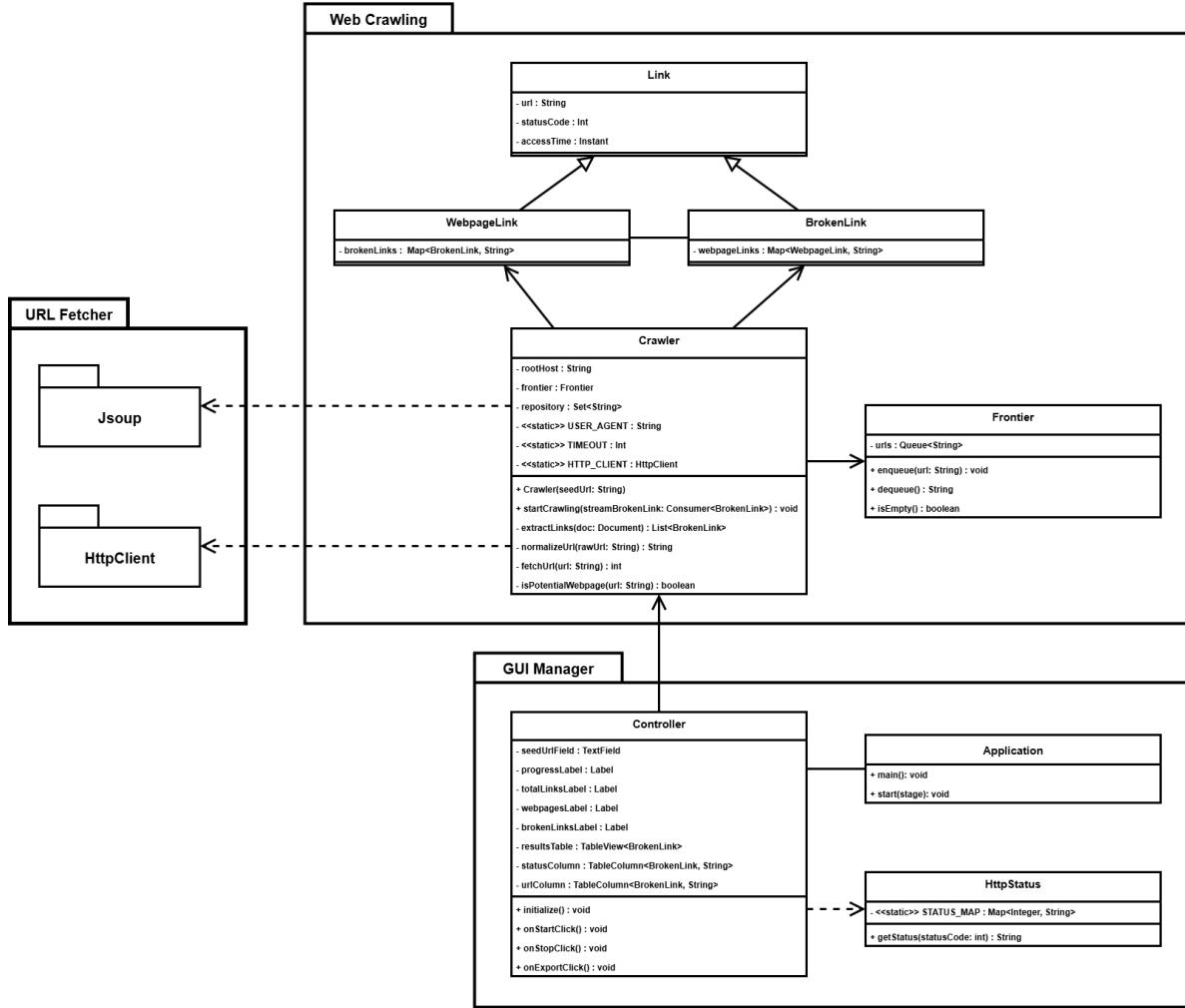
19 4.2 Perancangan Alur

20 Perancangan alur menggambarkan interaksi antarobjek pada sistem *Broken Link Checker* selama
21 proses pemeriksaan tautan berlangsung. Interaksi ini divisualisasikan dalam bentuk *sequence
22 diagram* yang menunjukkan urutan pesan antarobjek sejak pengguna memulai pemeriksaan hingga
23 proses selesai tanpa interupsi. Diagram lengkap ditunjukkan pada Gambar 4.2.

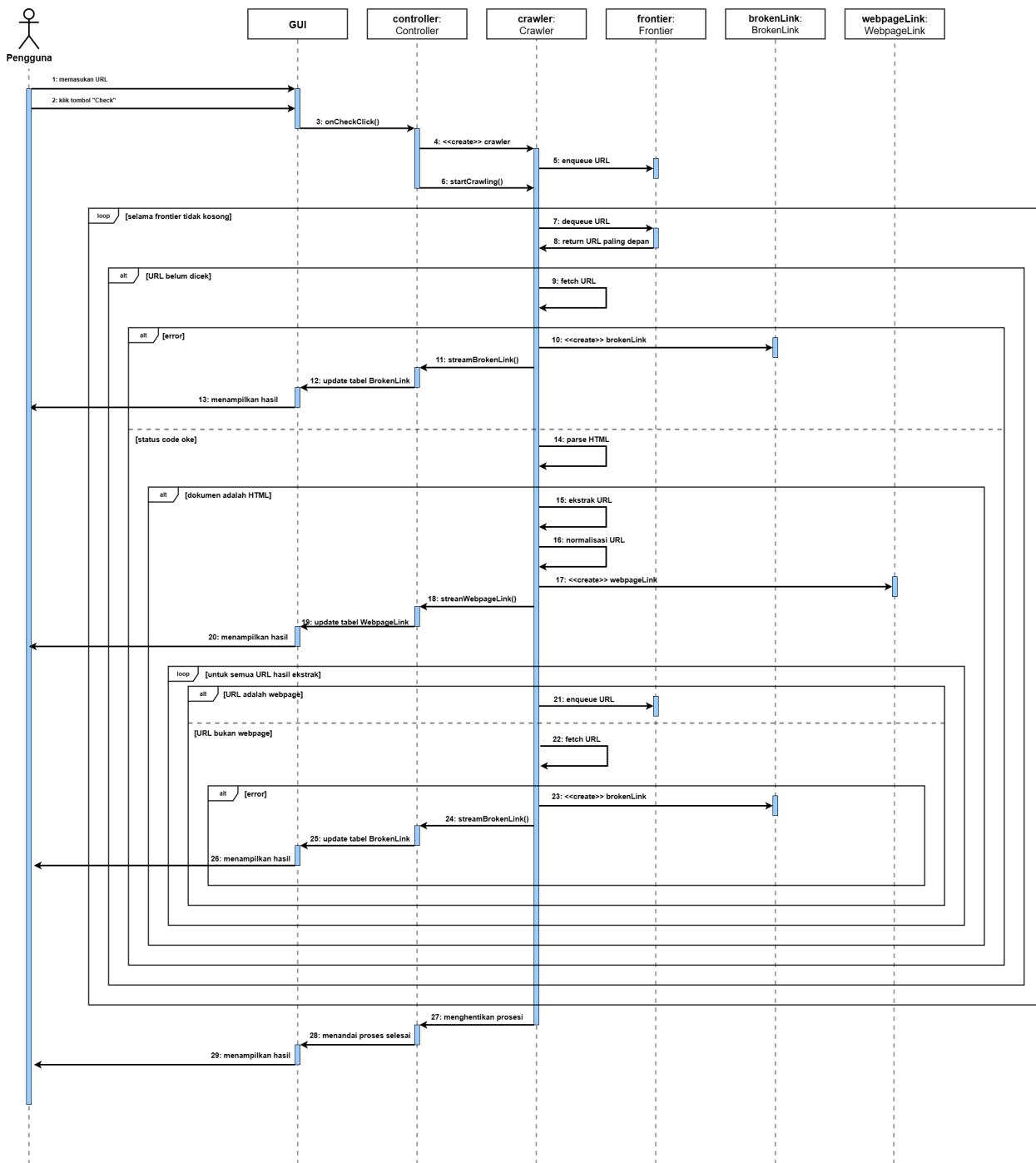
24 Berikut adalah penjelasan dari diagram pada Gambar 4.2:

- 25 1. (1) Pengguna memasukkan *seed URL* pada GUI.
- 26 2. (2) Pengguna menekan tombol *Check*.
- 27 3. (3) GUI memanggil metode `onCheckClick()` pada `Controller`.
- 28 4. (4) `Controller` membuat objek `Crawler` baru.
- 29 5. (5) `Crawler` menambahkan URL awal ke `Frontier` melalui `enqueueUrl()`.
- 30 6. (6) `Controller` memanggil `startCrawling()` pada `Crawler`.
- 31 7. (7–8) `Crawler` masuk ke dalam *loop* dengan mengambil URL dari `Frontier` menggunakan
32 `dequeue()`, yang mengembalikan URL paling depan.
- 33 8. (9) URL di-*fetch* oleh `Crawler`.
- 34 9. (10–12) Jika terjadi `error`, maka dibuat objek `BrokenLink`. Objek ini dikirim ke `Controller`
35 melalui `streamBrokenLink()`, kemudian (12–13) ditampilkan pada tabel `BrokenLink` di GUI.
- 36 10. (14–20) Jika status code hasil `fetch` oke, maka dilakukan:
 - 37 (a) (14) Parsing HTML.
 - 38 (b) (15–16) Ekstraksi dan normalisasi URL dari dokumen.
 - 39 (c) (17) Dibuat objek `WebpageLink`.
 - 40 (d) (18–20) Objek ini dikirim ke `Controller` melalui `streamWebpageLink()` dan ditampilkan
41 di tabel `WebpageLink` pada GUI.

- 1 11. (21–26) Untuk setiap URL hasil ekstraksi (*loop*):
 2 (a) Jika URL adalah halaman web, maka dimasukkan kembali ke **Frontier** melalui `enqueueUrl()`.
 3 (b) Jika URL bukan halaman web, dilakukan `fetch`.
 4 (c) (23–26) Jika `fetch` error, dibuat objek **BrokenLink**, dikirim ke **Controller** (`streamBrokenLink()`),
 5 kemudian diperbarui pada tabel dan ditampilkan di GUI.
 6 12. (27–29) Proses berulang sampai **Frontier** kosong. Pada akhir siklus, **Crawler** menghentikan
 7 proses, **Controller** menandai proses selesai, dan GUI menampilkan status akhir serta
 8 ringkasan hasil.



Gambar 4.1: Diagram kelas aplikasi



Gambar 4.2: Sequence diagram proses pemeriksaan tautan

4.3 Perancangan Antarmuka

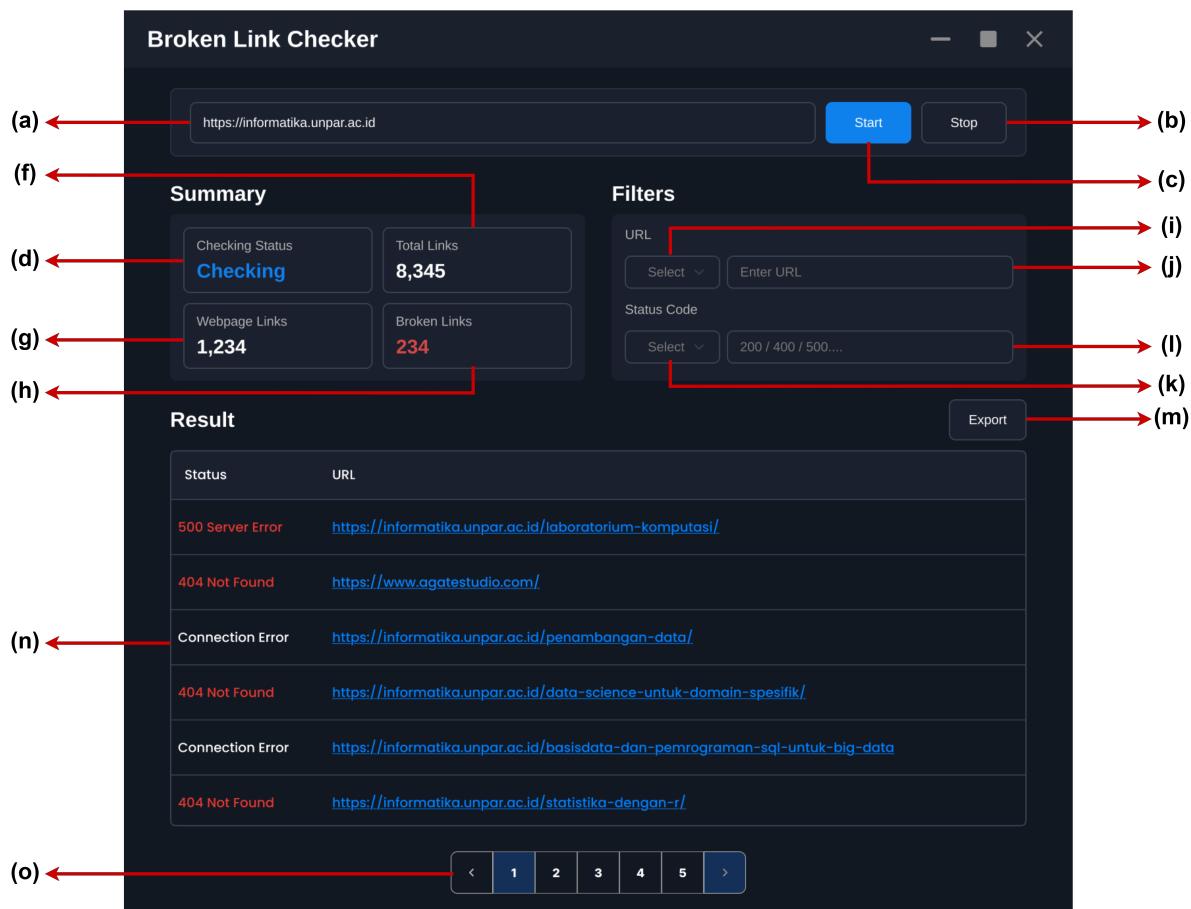
Perancangan antarmuka pengguna dibuat dalam bentuk desain fidelitas tinggi dengan tujuan untuk memberikan gambaran visual yang mendekati tampilan akhir aplikasi berbasis JavaFX. Perancangan ini mempertimbangkan aspek keterbacaan, konsistensi elemen antarmuka, serta kemudahan interaksi bagi pengguna dalam menjalankan proses pemeriksaan tautan rusak.

1. Jendela Utama

Jendela utama merupakan tampilan awal yang muncul ketika aplikasi dijalankan. Halaman ini

memfasilitasi pengguna untuk memasukkan URL situs web yang akan diperiksa, memulai atau menghentikan proses pemeriksaan, serta menampilkan hasil secara langsung. Elemen-elemen penting pada jendela ini ditunjukkan pada Gambar 4.3 dan dijelaskan sebagai berikut:

- (a) **Kolom Masukan URL:** digunakan untuk memasukkan alamat situs web yang akan diperiksa.
- (b) **Tombol Start:** digunakan untuk memulai proses pemeriksaan tautan.
- (c) **Tombol Stop:** menghentikan proses pemeriksaan yang sedang berlangsung.
- (d) **Checking Status:** menampilkan status terkini dari proses pemeriksaan, seperti *Checking*, *Stopped*, atau *Completed*.
- (e) **Total Links:** menunjukkan jumlah total tautan yang ditemukan selama proses *crawling*.
- (f) **Webpage Links:** menunjukkan jumlah tautan yang merupakan halaman situs web.
- (g) **Broken Links:** menunjukkan jumlah tautan rusak yang ditemukan selama proses pemeriksaan.
- (h) **Opsi Filter URL:** digunakan untuk menentukan jenis pencarian pada kolom URL. Opsi yang tersedia adalah:
 - **Equals:** menampilkan hasil dengan URL yang sama persis.
 - **Contains:** menampilkan hasil yang mengandung teks tertentu pada URL.
 - **Starts With:** menampilkan hasil dengan URL yang diawali teks tertentu.
 - **Ends With:** menampilkan hasil dengan URL yang diakhiri teks tertentu.
- (i) **Kolom Masukan Filter URL:** berfungsi untuk memasukkan kata kunci pencarian URL sesuai dengan opsi filter yang dipilih sebelumnya.
- (j) **Opsi Filter Status Code:** digunakan untuk menyaring hasil berdasarkan kategori kode status HTTP. Opsi yang tersedia adalah:
 - **Equals:** menampilkan tautan dengan kode status tertentu.
 - **Greater Than:** menampilkan tautan dengan kode status lebih besar dari nilai yang dimasukkan.
 - **Less Than:** menampilkan tautan dengan kode status lebih kecil dari nilai yang dimasukkan.
- (k) **Kolom Masukan Filter Status Code:** digunakan untuk memasukkan nilai numerik kode status HTTP sesuai dengan jenis filter yang dipilih pada poin sebelumnya.
- (l) **Tombol Export:** digunakan untuk mengekspor hasil pemeriksaan ke dalam berkas Excel.
- (m) **Tabel Hasil:** menampilkan daftar hasil pemeriksaan dalam bentuk tabel dengan dua kolom utama:
 - i. **Kolom Status:** menampilkan kode status HTTP dan *reason phrase*.
 - ii. **Kolom URL:** menampilkan alamat tautan rusak.
- (n) **Pagination:** digunakan untuk menavigasi hasil pemeriksaan jika jumlah tautan yang ditemukan cukup banyak.

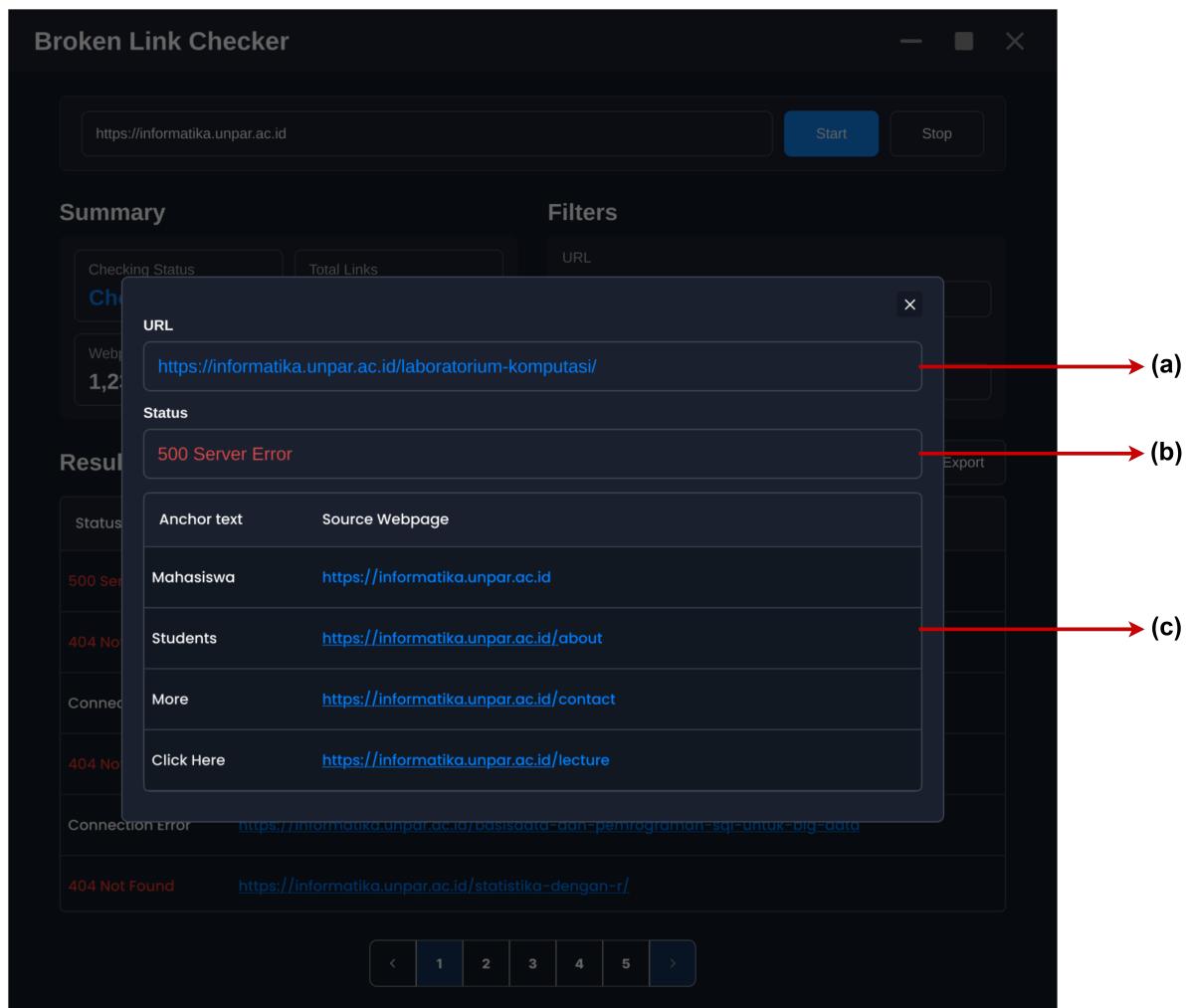


Gambar 4.3: Rancangan Halaman Utama

2. Jendela Detail Broken Link

Jendela ini muncul ketika pengguna memilih salah satu tautan rusak pada tabel hasil. Elemen-elemen penting pada jendela ini ditunjukkan pada Gambar 4.4 dan dijelaskan sebagai berikut:

- Kolom URL:** menampilkan alamat lengkap dari tautan rusak yang sedang diperiksa.
- Kolom Status:** menampilkan kode status HTTP dan *reason phrase*.
- Tabel Referensi Tautan:** berisi daftar halaman yang menjadi sumber dimana tautan rusak ditemukan, dengan dua kolom utama:
 - Anchor Text:** teks tautan yang muncul pada halaman sumber.
 - Source Webpage:** URL halaman tempat tautan rusak ditemukan.



Gambar 4.4: Rancangan Jendela Detail Broken Link

¹

BAB 5

²

IMPLEMENTASI DAN PENGUJIAN

- ³ Bab ini membahas hasil implementasi aplikasi desktop pemeriksa tautan rusak pada situs web
⁴ berdasarkan perancangan yang telah disusun pada Bab [4](#), yang mencakup penetapan lingkungan
⁵ implementasi, implementasi kelas-kelas utama dalam sistem, dan implementasi antarmuka pengguna.
⁶ Setelah implementasi, dilakukan pengujian fungsional dan eksperimental serta dilakukan analisis
⁷ terhadap hasil pengujian.

⁸ 5.1 Implementasi

⁹ 5.1.1 Lingkungan Implementasi

¹⁰ 5.1.2 Implementasi Kelas

¹¹ 5.1.3 Implementasi Antarmuka

¹² 5.2 Pengujian

¹³ 5.2.1 Pengujian Fungsional

¹⁴ 5.2.2 Pengujian Eksperimental

¹⁵ 5.2.3 Analisis Hasil Pengujian

¹

BAB 6

²

KESIMPULAN DAN SARAN

³ Bab ini menyajikan kesimpulan dari penelitian dan pengembangan aplikasi pemeriksa tautan
⁴ rusak yang telah dilakukan. Kesimpulan dirumuskan berdasarkan hasil analisis, perancangan,
⁵ implementasi, serta pengujian yang telah dipaparkan pada bab-bab sebelumnya. Selain itu, bab
⁶ ini juga memberikan saran yang dapat menjadi masukan untuk pengembangan lebih lanjut, baik
⁷ dalam peningkatan fitur maupun perluasan cakupan penelitian di masa mendatang.

⁸ 6.1 Kesimpulan

⁹ 6.2 Saran

DAFTAR REFERENSI

- [1] Fielding, R. T., Nottingham, M., dan Reschke, J. (2022) Http semantics. RFC 9110. RFC Editor, <http://www.rfc-editor.org>.
- [2] Berners-Lee, T., Fielding, R. T., dan Masinter, L. (2005) Uniform resource identifier (uri): Generic syntax. RFC 3986. RFC Editor, <http://www.rfc-editor.org>.
- [3] Powell, T. A. (2010) *HTML & CSS: The Complete Reference*, 5th edition. McGraw-Hill, New York.
- [4] Liu, B. (2011) *Web Data Mining: Exploring Hyperlinks, Contents, and Usage Data*, 2nd edition. Springer, Berlin, Heidelberg.
- [5] Sharan, K. dan Späth, P. (2022) *Learn JavaFX 17: Building User Experience and Interfaces with Java*, 2nd edition. Apress, Montgomery, AL, USA.
- [6] Faizi, A., Carter, S. R., dan Jensen, D. (1999) Http extensions for distributed authoring. RFC 2518. RFC Editor, <http://www.rfc-editor.org>.
- [7] Oku, K. (2017) An http status code for indicating hints. RFC 8297. RFC Editor, <http://www.rfc-editor.org>.
- [8] Dusseault, L. (2007) Http extensions for web distributed authoring and versioning (webdav). RFC 4918. RFC Editor, <http://www.rfc-editor.org>.
- [9] Goland, Y. Y., van Hoff, A., dan Hellerstein, D. M. (2002) Delta encoding in http. RFC 3229. RFC Editor, <http://www.rfc-editor.org>.
- [10] Thomson, M., Nottingham, M., dan Tarreau, W. (2018) Using early data in http. RFC 8470. RFC Editor, <http://www.rfc-editor.org>.
- [11] Fielding, R. T. dan Nottingham, M. (2012) Additional http status codes. RFC 6585. RFC Editor, <http://www.rfc-editor.org>.
- [12] Bray, T. (2016) An http status code to report legal obstacles. RFC 7725. RFC Editor, <http://www.rfc-editor.org>.
- [13] Holtzman, K. dan Mutz, A. (1998) Transparent content negotiation in http. RFC 2295. RFC Editor, <http://www.rfc-editor.org>.
- [14] Clemm, G., Crawford, J., Reschke, J. F., dan Whitehead, J. (2010) Binding extensions to web distributed authoring and versioning (webdav). RFC 5842. RFC Editor, <http://www.rfc-editor.org>.

LAMPIRAN A
KODE PROGRAM

LAMPIRAN B
HASIL EKSPERIMEN