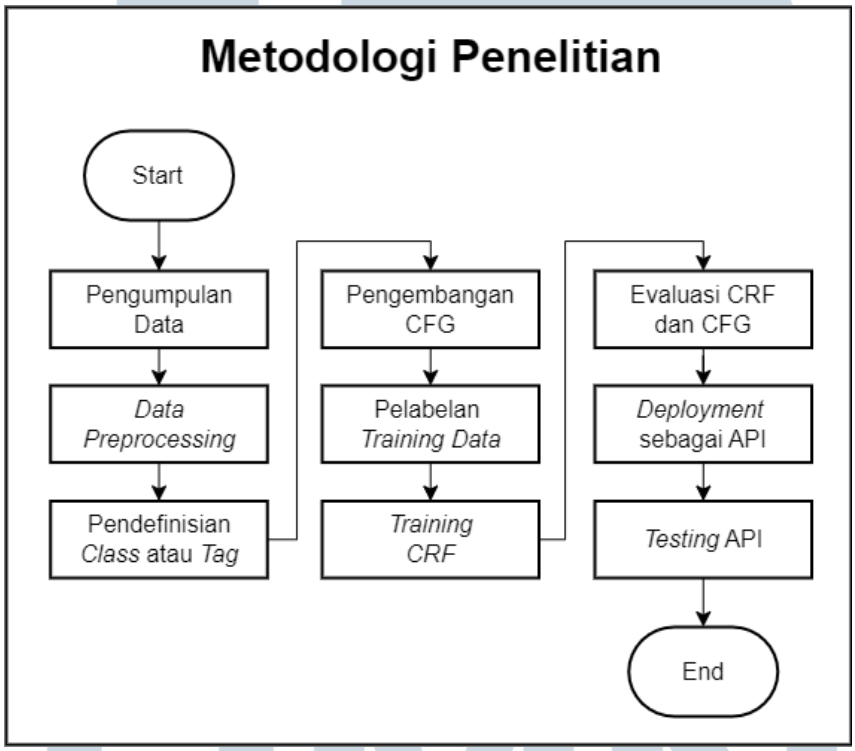
BAB 3
METODOLOGI PENELITIAN

Diagram alur dari metodologi/prosedur penelitian ”Algoritma U-Tapis Pendeteksi Kesalahan Sintaksis Kalimat Bahasa Indonesia” dapat dilihat dalam Gambar 3.1.



Gambar 3.1. Diagram alir *metodologi penelitian*

Berikut adalah uraian dari tiap tahapan metodologi penelitian ini:

3.1 Pengumpulan Data

Pengumpulan data dilakukan untuk mendapatkan *dataset* yang akan digunakan sebagai *training dataset* algoritma *Conditional Random Field* dan referensi untuk *rule-rule* algoritma *Context Free Grammar*. Data yang dikumpulkan adalah artikel-artikel berita yang telah dipublikasikan oleh media berita Tribun News. Penulis juga membuat data-data kalimat *training dataset* sendiri untuk memastikan semua kategori kata tercakup di dalam *training dataset*.

Training data akan menggunakan 60 artikel berita Tribun News dan data

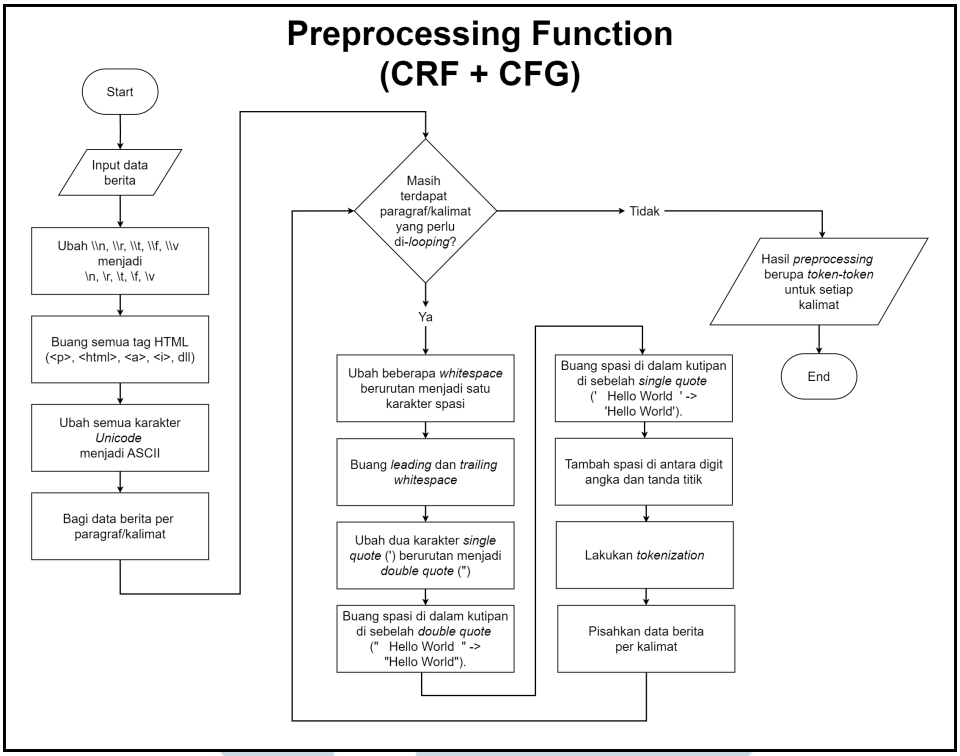
buatan penulis. *Testing data* akan menggunakan 15 artikel berita Tribun News. Rasio *train-test split* yang diharapkan dalam penelitian ini adalah 80% : 20%.

3.2 Data Preprocessing

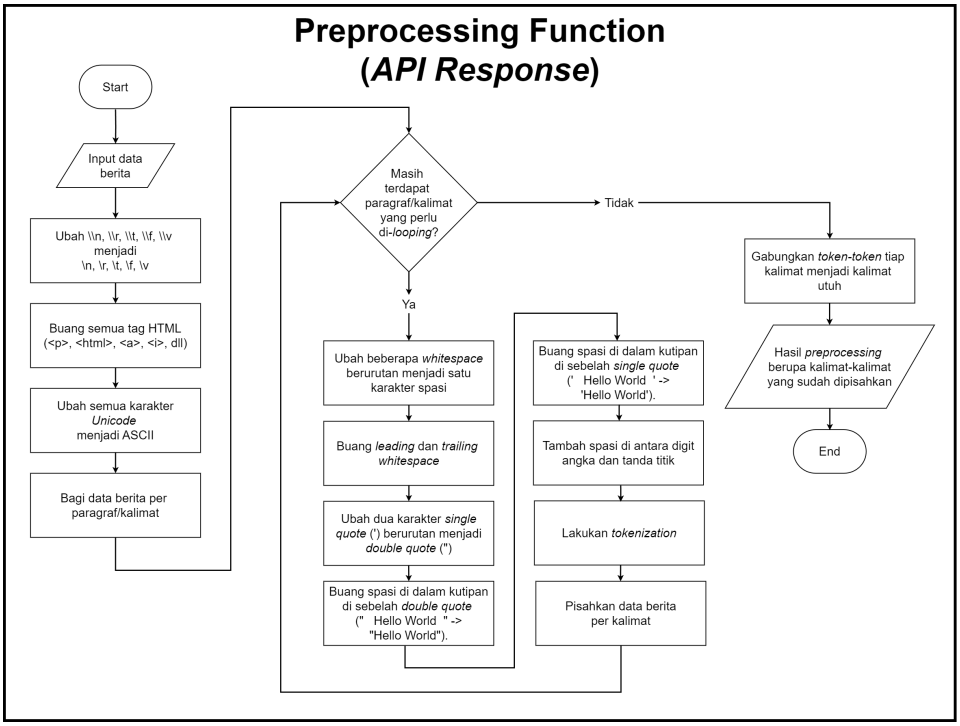
Data-data berita yang telah dikumpulkan akan melalui proses *data preprocessing* agar data berita tersebut siap digunakan untuk pengembangan algoritma *Conditional Random Field* dan pengembangan algoritma *Context Free Grammar*. Fungsi yang telah dibuat untuk *data preprocessing* ini akan digunakan sebagai fungsi *preprocessing* untuk setiap *input* yang diberikan ke ”Algoritma U-Tapis Pendeteksi Kesalahan Sintaksis Kalimat”.

Di dalam penelitian ini, terdapat dua jenis fungsi *preprocessing*, yaitu *CRF and CFG preprocessing function* dan *API preprocessing function*. *CRF and CFG preprocessing function* digunakan untuk memastikan masukan yang diterima algoritma siap digunakan oleh kedua algoritma tersebut, sedangkan fungsi *API preprocessing function* digunakan untuk mengelola keluaran kalimat yang akan dikembalikan sebagai *API response*. *CRF and CFG preprocessing function* akan membagi data artikel menjadi *token-token* per kalimat di dalam artikel tersebut. *API preprocessing function* akan membagi data artikel menjadi *string-string* per kalimat (*token-token* dari tiap kalimat akan digabungkan kembali menjadi kalimat utuh).

Diagram alir dari *CRF and CFG preprocessing function* dapat dilihat pada Gambar 3.2, sedangkan fungsi *preprocessing* untuk API dapat dilihat pada Gambar 3.3.



Gambar 3.2. Diagram alir *preprocessing function* untuk CFG dan CRF



Gambar 3.3. Diagram alir *preprocessing function* untuk API

Berikut adalah proses-proses yang dilakukan di dalam fungsi *preprocessing* untuk CRF dan CFG:

1. **Menghapus semua karakter *backslash* (\) dari karakter** `\\f, \\v, \\n, \\t, \\r, \\v` (*whitespace character*)

   Saat *Flask API framework* menerima suatu *request*, *framework* ini akan menambahkan *backslash* (\) untuk setiap karakter `\f, \v, \n, \t, \r, \v` (*whitespace character*). Proses ini akan mengganggu fungsionalitas dari kode-kode setelah kode ini. Oleh karena itu, karakter `\\f, \\v, \\n, \\t, \\r, \\v` harus dikembalikan menjadi karakter `\f, \v, \n, \t, \r, \v`).

2. **Menghapus semua elemen HTML**

   Teks data yang diterima oleh API dapat mengandung elemen HTML seperti `<body>, <head>, <div>, <a>, <br>, <script>, <link>, <p>`, dan sebagainya. *HTML tag* ini harus dihapus karena dapat mengganggu proses *part-of-speech tagging* CRF dan *parsing* CFG.

3. **Mengkonversi karakter-karakter dengan format unicode menjadi format ASCII**

   Semua karakter dengan format unicode harus diubah menjadi format ASCII yang memiliki arti yang sama. Proses ini dilakukan untuk memastikan tidak ada karakter-karakter yang memiliki arti yang sama, tetapi memiliki karakter yang berbeda. Contoh dari kasus ini, yaitu karakter kutip dua yang memiliki tiga versi berbeda (“ ” "). Ketiga karakter ini akan dianggap sebagai simbol yang berbeda meski memiliki arti yang sama.

4. **Memecah data berita menjadi paragraf-paragraf/kalimat-kalimat dengan karakter** `\n` **sebagai pemisah**

   Data berita yang diterima oleh API akan dipecah menjadi paragraf-paragraf atau kalimat-kalimat. Hasil dari pemecahan ini masih harus dikelola lebih lanjut untuk memastikan bahwa data yang telah dipecah ini benar-benar data kalimat dan bukan paragraf.

5. **Melakukan iterasi untuk melakukan penyesuaian lebih lanjut dan pemecahan paragraf menjadi kalimat**

Hasil dari proses pemecahan data berita menjadi kalimat-kalimat dan paragraf-paragraf akan ditindaklanjuti dengan pembuangan karakter-karakter yang tidak relevan, penambahan karakter lainnya bila dibutuhkan, dan proses validasi agar tiap hasil pecahan tersebut merupakan kalimat dan bukan paragraf.

Berikut adalah langkah-langkah kerja yang dilakukan pada tiap iterasi:

(a) **Menggantikan semua karakter *whitespace* dengan satu karakter spasi**

Proses ini bertujuan untuk memastikan tidak ada *whitespace* selain tanda spasi yang berjumlah satu karakter.

(b) **Membuang semua spasi di depan dan belakang pecahan data berita tersebut**

Proses ini bertujuan untuk memastikan tidak ada karakter spasi di depan atau di belakang pecahan data berita tersebut.

(c) **Mengubah dua tanda kutip tunggal (') yang bersebelahan menjadi tanda kutip ganda (")**

Proses ini bertujuan untuk memastikan tidak ada kutipan yang menggunakan dua tanda kutip tunggal. Semua kutipan harus menggunakan tanda kutip ganda.

(d) **Membuang whitespace di dalam kutipan di sebelah tanda double quote (")**

Kode *tokenization* akan memberikan *output* yang tidak terprediksi bila terdapat *whitespace* di dalam tanda kutipan. Oleh karena itu, semua *whitespace* yang berada di dalam kutipan dan menempel dengan tanda *double quote* (") akan dihapus (`"  Hello World  "` → `"Hello World"`).

(e) **Membuang whitespace di dalam kutipan di sebelah tanda single quote (')**

Kode *tokenization* akan memberikan *output* yang tidak terprediksi bila terdapat *whitespace* di dalam tanda kutipan. Oleh karena itu, semua *whitespace* yang berada di dalam kutipan dan menempel dengan tanda *single quote* (') akan dihapus (`'  Hello World  '` → `'Hello World'`).

(f) **Memisahkan tanda angka dan tanda titik di akhir angka tersebut**

Saat proses *tokenization* dilakukan, tanda titik yang menempel dengan angka akan digolongkan sebagai satu kesatuan kata. Hal ini dapat menyebabkan kesalahan pada proses segmentasi/pemisahan kalimat di tahap selanjutnya. Untuk menghindari itu, tanda titik yang berada di akhir angka akan dipisahkan dengan tanda spasi (1234. $\rightarrow$ 1234 .). Tanda titik yang diapit oleh dua angka dan tanda titik yang berada di depan angka tidak akan dipisahkan dengan tanda spasi.

(g) **Melakukan tokenization**

Proses *tokenization* dilakukan pada setiap *input* paragraf/kalimat tersebut. Hasil *tokenization* ini adalah data paragraf/kalimat yang telah dipisahkan per kata atau simbol.

(h) **Melakukan segmentasi kalimat**

Proses segmentasi kalimat dilakukan untuk memecah lebih lanjut paragraf atau kalimat hasil pemecahan sebelumnya menjadi kata-kata atau simbol-simbol (*tokenization*). Setelah proses *tokenization* dilakukan, proses *sentence segmentation* dilakukan untuk memastikan semua data pecahan tersebut adalah data kalimat dan bukan paragraf. *token-token* tersebut akan dianggap sebuah kesatuan kalimat bila ditemukan tanda akhir kalimat (`.?!`) baik di dalam kutipan maupun di luar kutipan. Semua karakter selain tanda koma dan tanda akhir kalimat (`,.?!`) akan dibuang karena klausa di dalam kutipan bersifat non-formal dan tidak dapat diuji kebenaran sintaksisnya.

Hasil akhir dari *CRF and CFG preprocessing function* adalah data teks yang telah dibersihkan dan dipisahkan per kalimat. Tiap kalimat tersebut juga telah dipecah menjadi *tag-tag* per kata/simbol. *Output* yang diberikan oleh fungsi ini adalah *list of list of strings* (*list(list(str))*).

API preprocessing function memiliki proses yang sama dengan *CRF and CFG preprocessing function*. Perbedaan yang dimilikinya, yaitu adanya proses penggabungan *token-token* tiap kalimat menjadi satu kalimat utuh kembali. *Output* yang diberikan oleh fungsi ini adalah *list of strings* (*list(str)*).

3.3 Pendefinisian *Class* atau Tag

Setelah proses *preprocessing* selesai dilakukan, proses pendefinisian *class* dilakukan. Tahap pendefinisian *class* atau *tag* dilakukan untuk menentukan *class* yang digunakan saat proses *part-of-speech tagging* dan proses *parsing*. *Class-class* ini harus ditentukan sebelum proses pelabelan *training data* dan pendefinisian *rule-rule* untuk *parsing* dapat dilakukan.

Pada awal mula proses pendefinisian *Class-class* ini, *class-class* yang dibuat hanya mencakup jenis-jenis kata yang umum ada pada bahasa Indonesia sesuai kaidah bahasa Indonesia yang baik dan benar (nomina, verba, adjektiva, adverbia, konjungsi, dan sebagainya). Seiring pengembangan penelitian ini, *class-class* baru akan ditambahkan sesuai dengan kebutuhan algoritma *Context Free Grammar* untuk kata-kata yang membutuhkan perlakuan khusus (kata pengingkaran 'tidak' dan 'belum', konjungsi korelatif, kata pengingkaran 'bukan' yang hanya bisa diikuti oleh nomina, dan sebagainya).

3.4 Pengembangan Algoritma Context Free Grammar

Setelah *class* atau *tag* telah ditentukan, tahap perancangan *rule-rule* untuk *Context Free Grammar* dimulai. Tahap ini dilakukan sebelum tahap pelabelan data untuk mengantisipasi perubahan *class* atau *tag* yang perlu dilakukan untuk mengakomodasi kebutuhan *parsing* oleh *Context Free Grammar*. Perancangan *rule-rule* ini dimulai dengan menggunakan aturan-aturan tata baku bahasa Indonesia yang baik dan benar [18, 19]. Kemudian, *rule-rule* ini akan diubah atau *rule-rule* baru akan ditambahkan sesuai dengan kebutuhan performa dan cara kerja algoritma tersebut (menghindari *infinite recursion* dan sebagainya).

Selain mengembangkan *rule-rule Context Free Grammar*, pengembangan algoritma *Context Free Grammar* ini juga mencakup pengembangan metode *parsing* yang optimal. Di dalam penelitian ini, algoritma *Context Free Grammar* akan menggunakan teknik *left-corner parsing* untuk meningkatkan kecepatan *parsing*. Algoritma *Context Free Grammar* juga akan menggunakan teknik *early stopping* untuk menghentikan proses *parsing* bila ditemukan minimal satu *parse tree* (hal ini karena kebenaran sintaksis suatu kalimat ditentukan dengan adanya minimal satu *parse tree* sehingga tidak perlu mencari semua kombinasi *parse tree*). Kode pengembangan ini dapat ditemukan di *Google Colab* penelitian ini (https://colab.research.google.com/drive/1F7xqFX6OBHFEL3dk-

Oc2_B1GKIfLIgcD?usp=sharing) dan *GitHub repository Context Free Grammar* (https://github.com/denn-acrymoore/utapis-cfg-files).

Setelah itu, algoritma *Context Free Grammar* ini akan diujikan dengan beberapa kalimat yang memiliki struktur khusus (kalimat majemuk bertingkat, kalimat dengan suplementasi, kalimat dengan pelengkap klausa, dan sebagainya). Proses pengujian ini dapat dilihat pada *Google Colab* penelitian ini (https://colab.research.google.com/drive/1F7xqFX6OBHFEL3dk-Oc2_B1GKIfLIgcD?usp=sharing) dan pada *unit test* di *GitHub repository Context Free Grammar* (https://github.com/denn-acrymoore/utapis-cfg-files). Bila ditemukan kalimat-kalimat yang memiliki hasil pengecekan sintaksis yang salah, *rule-rule Context Free Grammar* akan diperbaiki untuk mencakup kalimat tersebut.

3.5 Pelabelan Training Data

Setelah *rule-rule* untuk *Context Free Grammar* telah dibuat dan *class-class* yang akan digunakan telah difinalisasikan, proses pelabelan *training data* dimulai. Proses pelabelan ini dilakukan secara manual oleh penulis pada seluruh *token* (kata atau simbol) dari setiap kalimat berita yang digunakan sebagai *training data* algoritma *Conditional Random Field*. *Training data* yang digunakan adalah data-data artikel berita Tribun News dan data-data kalimat tambahan buatan penulis. Data-data kalimat buatan penulis digunakan untuk memastikan bahwa semua kata-kata yang jarang digunakan di dalam kalimat berita telah tercakup ke dalam *training data*. Proses pelabelan dilakukan sesuai dengan tata baku bahasa Indonesia yang baik dan benar [18, 19].

3.6 Training Conditional Random Field

Saat proses pelabelan *training data* selesai dilakukan, proses *training* model *Conditional Random Field (CRF)* dilakukan. Model *Conditional Random Field* akan diinisialisasikan dengan *custom feature function* sebelum proses *training* dilakukan. Berikut adalah *feature-feature* yang akan diekstraksi oleh *custom feature function* tersebut:

1. *Current word.*
2. *Previous word (if any).*
3. *Previous previous word (if any).*

4. *Next word (if any).*
5. *Next next word (if any).*
6. *Is the word capitalized?*
7. *Is the first word in the sentence?*
8. *Does it contain punctuation?*
9. *Does it contain a number?*
10. *Is the word all number (with or without ., between number)?*
11. *Is the word all uppercase?*
12. *Is the word all uppercase + symbol?*
13. *Prefixes up to length 4.*
14. *Suffixes up to length 4.*

Setelah proses *training selesai*, algoritma *Conditional Random Field* akan diuji dengan beberapa kalimat yang memiliki struktur khusus (kalimat dengan konjungsi koordinatif, kalimat dengan konjungsi subordinatif, kalimat dengan frasa nominal sebagai subjek, kalimat dengan kata pengingkaran 'tidak' dan 'bukan', dan sebagainya) untuk memastikan algoritma tersebut dapat memberikan prediksi yang tepat. Proses pengujian kalimat-kalimat dengan struktur khusus tersebut dapat dilihat pada *Google Colab* penelitian ini (https://colab.research.google.com/drive/1F7xqFX6OBHFEL3dk-Oc2_B1GKIfLIgcD?usp=sharing). Bila ditemukan kata-kata yang tidak dapat diprediksi dengan tepat oleh algoritma, kata-kata tersebut akan ditambahkan di dalam kalimat buatan penulis dan proses *training* akan dilakukan kembali.

Setelah hasil *training* sudah cukup memuaskan, model *Conditional Random Field* tersebut dihubungkan dengan algoritma *Context Free Grammar* agar *output* dari model *Conditional Random Field* dapat digunakan sebagai *input* dari *Context Free Grammar*.

3.7 Evaluasi Model Conditional Random Field (CRF) dan *Rule-Rule* Context Free Grammar (CFG)

Evaluasi model *Conditional Random Field* dan algoritma *Context Free Grammar* dilakukan untuk mengetahui performa dari masing-masing algoritma. Tahap evaluasi ini akan menggunakan 15 artikel berita Tribun News sebagai *test data*. Evaluasi ini akan dilakukan pada tiap algoritma secara terpisah dan pada kedua algoritma tersebut secara bersamaan (hasil prediksi *Conditional Random Field* menjadi *input* bagi *Context Free Grammar*). Evaluasi akan menggunakan metrik *precision, recall, accuracy, F1-Score/F-Measure*. Proses evaluasi ini juga akan menggunakan *macro-averaged* dan *weighted-averaged* untuk kalkulasi *precision, recall*, dan *F1-Score/F-Measure* karena algoritma *Conditional Random Field* memiliki lebih dari dua *class* (bukan *true-false*) dan sampel data kalimat berita untuk algoritma *Context Free Grammar* cenderung memiliki ketimpangan (jumlah kalimat dengan sintaksis yang benar lebih banyak dari kalimat dengan sintaksis yang salah). Bila hasil evaluasi tidak mencapai target yang diinginkan, *Conditional Random Field* dan *Context Free Grammar* akan diperbaiki. *Class-class/tag-tag* yang telah ditentukan juga dapat diubah bila diperlukan.

3.8 Deployment sebagai Application Programming Interface/API

Setelah proses pengembangan selesai, kedua algoritma itu akan diintegrasikan dengan *Flask Framework* di *Python* dalam bentuk *Application Programming Interface (API)*. Proses *deployment* dilakukan dalam bentuk API untuk memudahkan penggunaan dari berbagai media seperti *website*, aplikasi *Android*, dan sebagainya.

”Algoritma U-Tapis Pendeteksi Kesalahan Sintaksis Kalimat” akan di-*deploy* ke *endpoint* `http://{hostname_yang_digunakan}:{port_yang_digunakan}/utapis-cek-sintaksis-kal`. *Hostname* dan *port* yang digunakan dapat berubah sesuai dengan pengaturan pada *Python Flask Web Framework*. Untuk menggunakan API ini, pengguna harus mengirimkan *POST request* ke *endpoint*. Konten dari *POST request* tersebut adalah *form-element* ”*article*” yang berisi teks artikel berita yang ingin diuji. Setelah proses validasi sintaksis selesai, API akan mengirimkan *API response* dalam format JSON (*JavaScript Object Notation*). Format dari *JSON response* yang diterima dapat dilihat pada Kode 3.1.

```
{
    "results": [
        {
            "is_valid": boolean,
            "sentence": string
        },
        ...
    ]
}
```

Kode 3.1: *JSON API response*

API ini memiliki dua pengaturan opsional untuk mengatur *response* yang diberikan oleh API. Kedua pengaturan ini diberikan dalam bentuk *query parameter*, yaitu `bool-only` (`http://{hostname}:{port}/utapis-cek-sintaksis-kal?bool-only=1`) dan `false-only` (`http://{hostname}:{port}/utapis-cek-sintaksis-kal?false-only=1`). Pengaturan `bool-only` menyebabkan *API response* hanya mengembalikan *list-of-boolean* untuk setiap data kalimat berita yang diberikan. Pengaturan `false-only` menyebabkan *API response* mengembalikan *list-of-string* berupa kalimat-kalimat berita dengan sintaksis yang salah. Format dari *JSON response* untuk `bool-only` dan `false-only` telah diuraikan pada Kode 3.2 dan Kode 3.3.

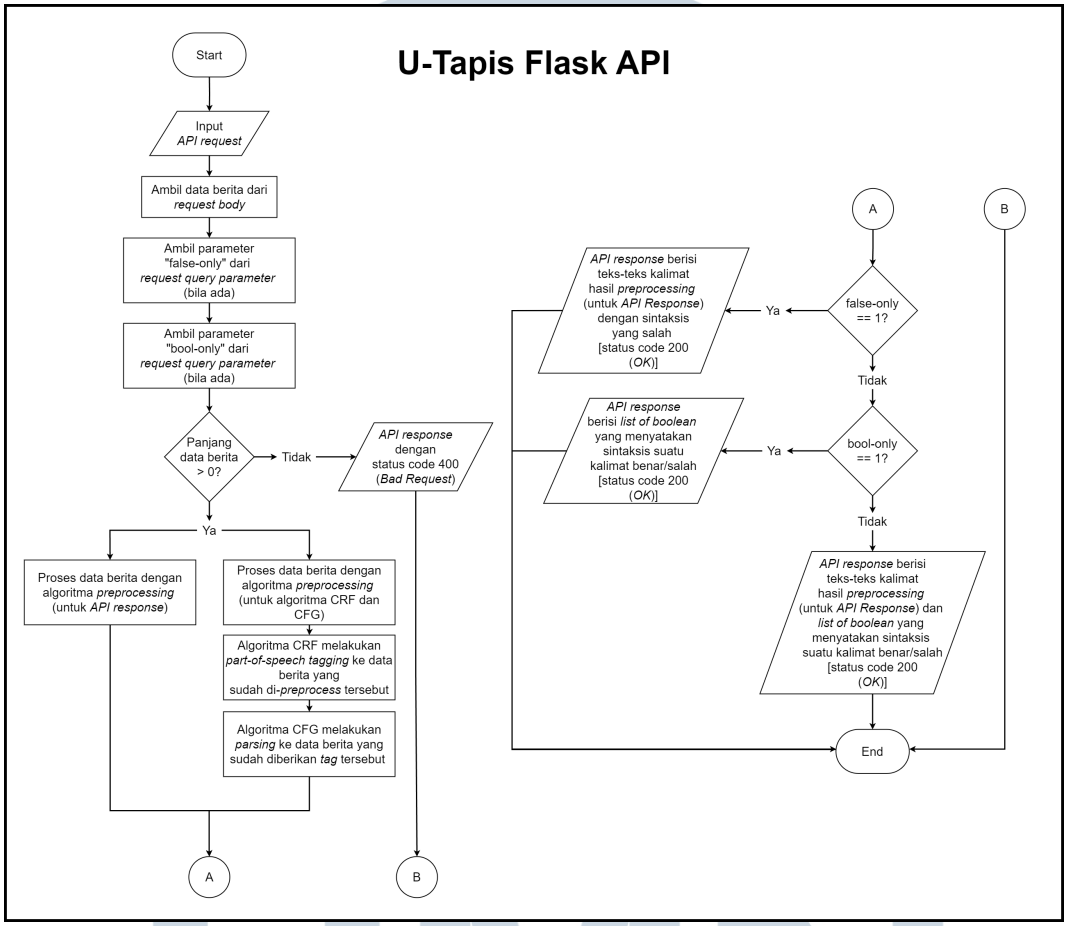
```
{
    "results": [
        {
            "is_valid": boolean
        },
        ...
    ]
}
```

Kode 3.2: *Bool-Only JSON API response*

```
{
    "results": [
        {
            "sentence": string
        },
        ...
    ]
}
```

Kode 3.3: *False-Only JSON API response*

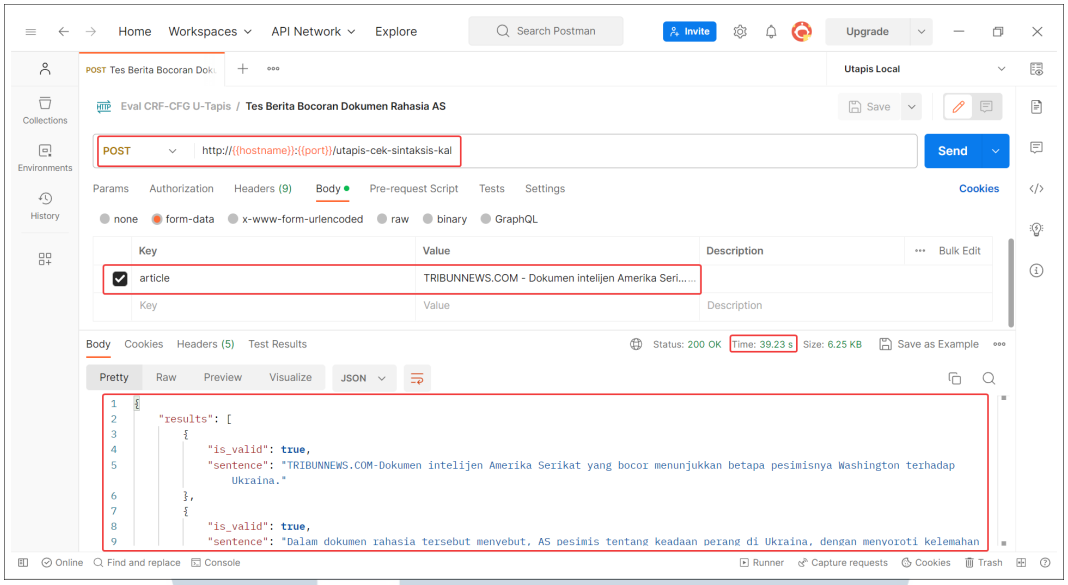
Diagram alir dari ”API U-Tapis Pendeteksi Kesalahan Sintaksis Kalimat” dapat dilihat pada Gambar 3.4.



Gambar 3.4. Flowchart *Application Programming Interface* (API) dari U-Tapis Pendeteksi Kesalahan Sintaksis Kalimat

3.9 Testing Application Programming Interface/API

Setelah proses *deployment* API selesai dilakukan, proses pengujian API dimulai. Pengujian ini dilakukan untuk memastikan API dapat menjalankan fungsionalitas ”Algoritma U-Tapis Pendeteksi Kesalahan Sintaksis Kalimat” tanpa kendala. Proses pengujian *run time* dari ”Algoritma U-Tapis Pendeteksi Kesalahan Sintaksis Kalimat” juga dilakukan pada tahap ini dengan menggunakan aplikasi Postman dengan menggunakan 15 artikel berita Tribun News sebagai *test data*. Tampilan dari aplikasi Postman dapat dilihat pada Gambar 3.5.



Gambar 3.5. Tampilan aplikasi Postman