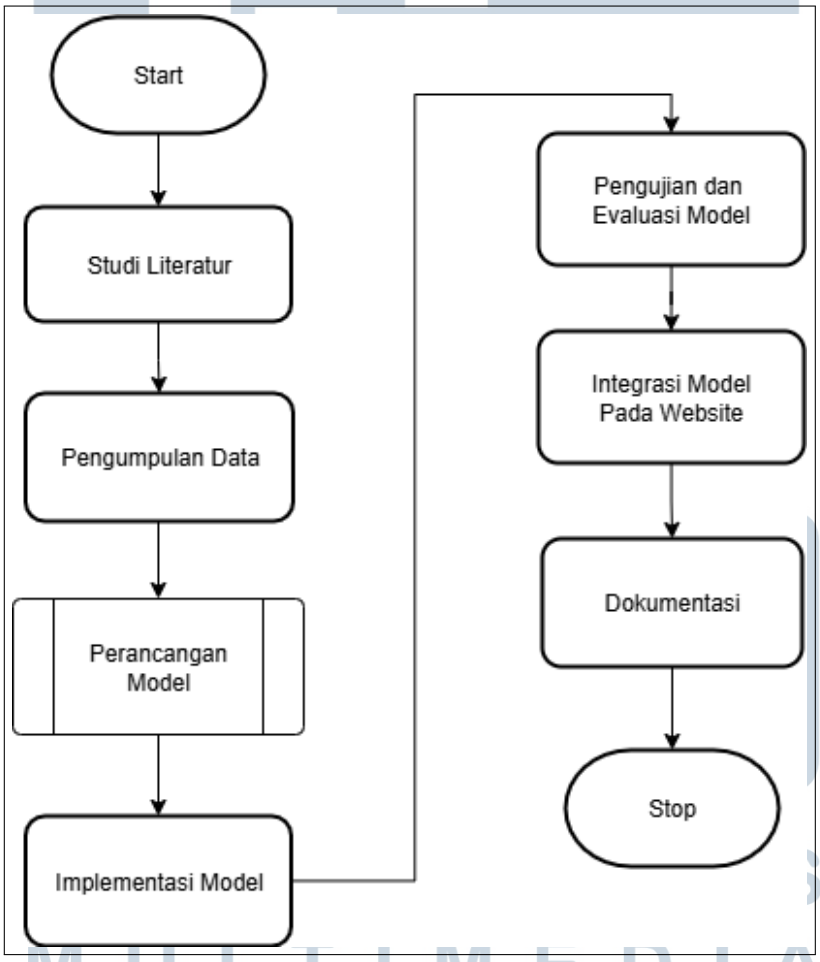
BAB 3
METODOLOGI PENELITIAN

3.1 Metodologi Penelitian

Penelitian berjudul Implementasi Algoritma Naive Bayes untuk Klasifikasi Teks Phishing dilakukan dengan bertahap. Pada bagian ini akan dijabarkan tahap-tahap yang akan dilakukan dalam menyusun dan mengerjakan penelitian, seperti Studi Literatur hingga Dokumentasi. Berikut metodologi yang digunakan:



Gambar 3.1. Flowchart Metodologi Penelitan

1. Studi Literatur

   Pada tahap awal penelitian, studi literatur dilakukan untuk mengumpulkan informasi berdasarkan penelitian yang sudah dilakukan sebelumnya. Pada penelitian ini literatur yang digunakan adalah literatur-literatur yang

berhubungan dengan *text classification*, *precision*, *recall*, *f1-score*, *accuracy*, algoritma *Naïve Bayes*, dan beberapa literatur lain yang berhubungan dengan penelitian ini. Literatur-literatur yang digunakan ini lalu dikumpulkan dan dijadikan referensi pada penelitian ini.

2. Pengumpulan Data

   Pada tahap pengumpulan data, peneliti mengumpulkan dataset untuk klasifikasi teks *phishing* dari penelitian sebelumnya, dan dari *website* Kaggle. Dataset yang dikumpulkan adalah dataset yang berisi data teks *phishing* dan memiliki label.

3. Perancangan Model

   Pada tahapan ini, peneliti melakukan perancangan terhadap model untuk klasifikasi teks dengan algoritma *Naïve Bayes*.

4. Implementasi Model

   Pada tahap implementasi model, peneliti melakukan pembuatan model sesuai yang sudah dirancang di tahap sebelumnya. Pembuatan model dimulai dari memuat *dataset*, *labelling* data, *pre-processing*. *training model*, dan *evaluasi model*.

5. Pengujian dan Evaluasi Model

   Pada tahap ini, model yang sudah di buat di tahapan sebelumnya akan melalui pengujian dengan beberapa skenario dengan tujuan untuk mencari model dengan tingkat *precision*, *recall*, *f1-score*, *accuracy* yang terbaik. Setelah pengujian selesai, dilakukan evaluasi untuk merangkum apa saja yang telah dilakukan di tahap pengujian.

6. Integrasi Model pada Website

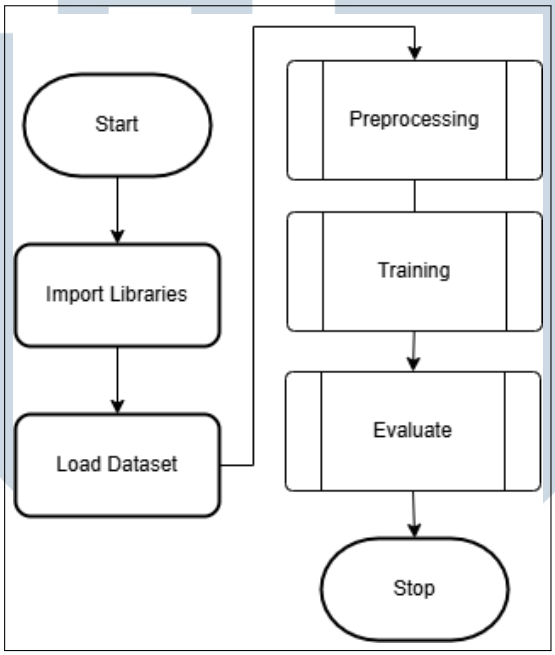
   Setelah model selesai diuji dan dievaluasi, langkah selanjutnya adalah mengintegrasikan model klasifikasi ke dalam sebuah *website*. Proses integrasi ini mencakup pembuatan *frontend* yaitu antarmuka pengguna untuk memasukkan teks, dan *backend* yaitu pemrosesan teks melalui model *Naïve Bayes*.

7. Dokumentasi

   Tahap dokumentasi merupakan tahap terakhir, yaitu dimana peneliti menuliskan dokumentasi tahap-tahap pembuatan sistem dalam laporan yang terstruktur.

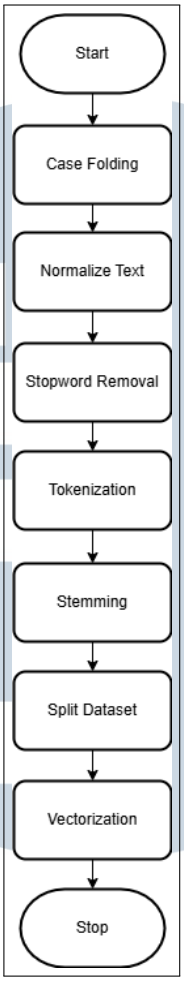
3.2 Perancangan Model

Pada bagian ini, dijelaskan tahap-tahap yang dilakukan ketika merancang model klasifikasi teks *phishing* dengan algoritma *Naïve Bayes*. Gambar 3.2 adalah *flowchart* perancangan model secara keseluruhan.



Gambar 3.2. Flowchart Utama

1. *Import Libraries*
   Tahap *import libraries* ini merupakan tahap awal dalam perancangan model, yaitu meng-import *libraries* yang diperlukan selama proses pengerjaan model, *libraries* yang di import membantu memudahkan pengerjaan model seperti di tahap *preprocessing*, *training*, dan *evaluation* karena memanfaatkan fungsi-fungsi dari *library* itu.
2. *Load Dataset*
   Pada tahap *load dataset*, *dataset* yang sudah disiapkan di load kedalam *workspace* untuk persiapan *preprocessing*.
3. *Preprocessing*
   Tahap *preprocessing* adalah tahap dimana *dataset* akan dibersihkan terlebih dahulu sebelum digunakan untuk membuat model.



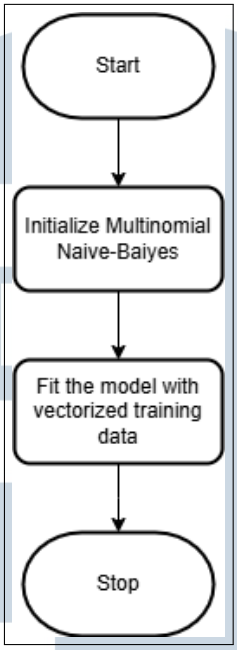
Gambar 3.3. Flowchart *Preprocessing*

Gambar 3.3 adalah *flowchart preprocessing* yang dimulai dengan *case folding* yang mengubah kata menjadi huruf kecil semua, menghapus angka, dan menghapus karakter tanda baca dari data. Pada tahap *normalize text* bertujuan untuk menyamakan bentuk setiap kata agar lebih konsisten dan mudah diproses. Berikutnya pada tahap *stopword removal* yang berguna untuk meningkatkan kualitas dari analisa klasifikasi teks dengan cara mengurangi dimensi data. Lalu berikutnya adalah tahap *tokenization*, *tokenization* dilakukan untuk mengubah kalimat menjadi satuan kata dalam *array*, setelah dilakukan *tokenization* baru data dapat dilakukan proses berikutnya yaitu *stemming* yang berguna untuk menghapus imbuhan pada awal dan akhir suatu kata sehingga tersisa kata dasarnya saja.

4. *Training*

   Setelah *dataset* dibersihkan, maka *dataset* sudah siap digunakan untuk proses

training model Multinomial *Naïve Bayes* untuk klasifikasi teks.

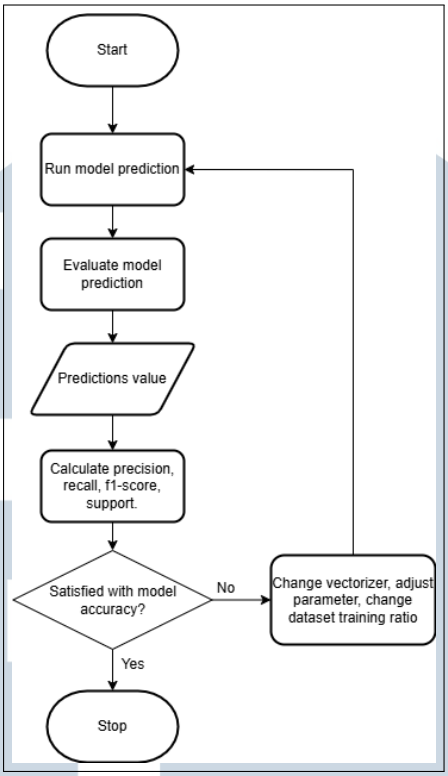


Gambar 3.4. Flowchart *Training*

Gambar 3.4 adalah *flowchart training* yang dimulai dengan inisialisasi model Multinomial *Naïve Bayes*, lalu model dilakukan *fitting* dengan *dataset* training yang sudah disiapkan.

5. *Evaluate*

   Pada tahap ini, model yang sudah di buat di tahapan sebelumnya akan dilakukan uji dengan *dataset testing*, dan evaluasi untuk mengukur kinerja model yang diukur dari tingkat *precision*, *recall*, *f1-score*, *accuracy*. *Precision* adalah ketepatan model dalam memprediksi kelas positif, *recall* merupakan ukuran sejauh mana model berhasil menemukan semua data positif, sedangkan *f1-score* adalah rata-rata harmonis antara *precision* dan *recall*, dan terakhir *accuracy* adalah tingkat kebenaran prediksi model baik positif maupun negatif.



Gambar 3.5. Flowchart *Evaluate*

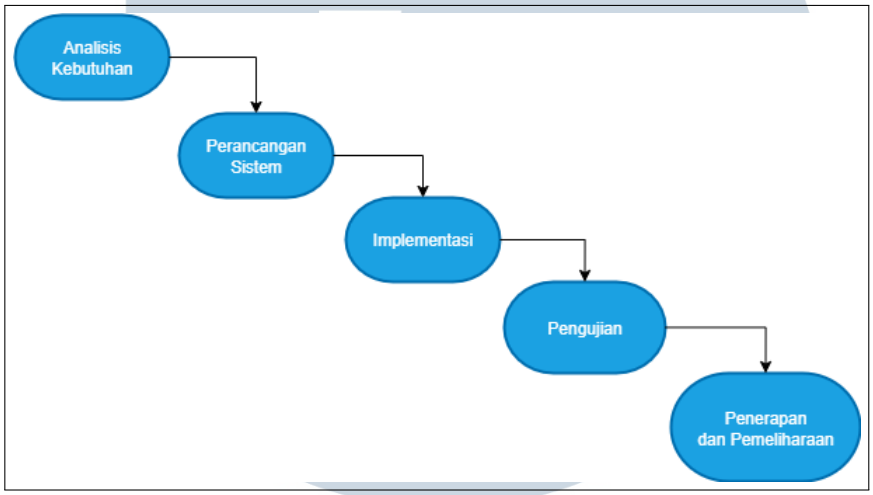
Gambar 3.5 adalah *flowchart evaluate* yang bertujuan untuk menilai dan mengevaluasi model yang telah dibuat dan memastikan model itu telah berjalan sesuai ekspektasi berdasarkan tingkat *precision*, *recall*, *f1-score*, *accuracy*. Jika model masih belum memuaskan dalam tingkat akurasi maka dapat dilakukan *training* data ulang dengan *vectorizer*, parameter model, dan rasio *dataset training* yang berbeda untuk mencapai hasil model yang terbaik.

3.3 Rancang Bangun Aplikasi

Setelah menyelesaikan perancangan model klasifikasi teks *phishing* dengan algoritma *Naïve Bayes* pada tahap sebelumnya, tahap berikutnya adalah mengimplementasikan model tersebut ke dalam sebuah aplikasi berbasis *website*. Tujuan implementasi ini adalah menciptakan tampilan antarmuka yang *user-friendly*, sehingga model yang telah dibuat dapat digunakan untuk melakukan klasifikasi teks. Aplikasi ini diberi nama *"TextCheckr"*.

3.3.1 Metode Pengembangan

Pengembangan aplikasi ini menggunakan metode *Waterfall*, yaitu metode pengembangan perangkat lunak yang bersifat sekuensial dan sistematis, di mana setiap tahapan harus diselesaikan terlebih dahulu sebelum melanjutkan ke tahap berikutnya. Tahapan-tahapan dalam metode *Waterfall* yang diterapkan pada pengembangan aplikasi *TextCheckr* dapat dilihat pada Gambar 3.6.



Gambar 3.6. Flowchart *Waterfall*

1. Analisis Kebutuhan
   Tahapan ini mencakup identifikasi kebutuhan sistem baik dari sisi *user interface* maupun kebutuhan fungsional dari sistem klasifikasi teks *phishing*. Kebutuhan yang diidentifikasi mencakup fungsi untuk memasukkan teks, memilih konfigurasi model, seperti *split data* dan jenis *vectorizer*, serta menampilkan hasil klasifikasi.

2. Perancangan Sistem
   Pada tahap ini dilakukan perancangan arsitektur aplikasi dan alur komunikasi antara komponen *frontend* dan *backend*. Desain *user interface* pengguna dirancang agar mudah digunakan dan informatif, sehingga pengguna dapat dengan mudah memasukkan teks, memilih konfigurasi model, dan melihat hasil klasifikasi. Untuk mendukung proses perancangan ini, dibuatlah *wireframe* yang menggambarkan tampilan halaman utama, form input teks, opsi konfigurasi model, tombol klasifikasi, serta area untuk menampilkan hasil klasifikasi.

3. Implementasi

   Tahapan ini melibatkan pembuatan aplikasi baik pada sisi *frontend* menggunakan *React.js* dan *TailwindCSS*, maupun pada sisi *backend* menggunakan *Flask*. Proses klasifikasi dilakukan di *back-end* dengan memuat model Naïve Bayes dan vectorizer berdasarkan input konfigurasi dari *frontend* lalu mengembalikan hasil klasifikasi ke *frontend*.

4. Pengujian

   Setelah implementasi selesai, dilakukan pengujian sistem untuk memastikan bahwa aplikasi dapat berjalan sesuai dengan fungsinya dan tanpa *bug*. Pengujian meliputi validasi hasil klasifikasi, pengujian antarmuka pengguna, serta pengujian komunikasi *API* antara *frontend* dan *backend*.

5. Penerapan dan Pemeliharaan

   Tahap ini dilakukan setelah aplikasi selesai diimplementasikan dan melalui proses pengujian. Langkah pertama dalam tahap ini adalah *deployment* atau penerapan, yaitu proses menempatkan aplikasi ke dalam lingkungan produksi agar dapat diakses dan digunakan oleh pengguna secara luas. Setelah aplikasi berhasil dideploy, dilakukan proses *maintenance* atau pemeliharaan yang mencakup perbaikan jika ditemukan *bug*, penyesuaian terhadap perubahan kebutuhan pengguna, serta optimalisasi performa aplikasi.

3.3.2 Arsitektur Aplikasi *"TextCheckr"*

Pengembangan aplikasi *"TextCheckr"* menggunakan arsitektur *Client-Server*, yaitu arsitektur di mana tampilan untuk klien (*front-end*) seperti *form*, hasil klasifikasi teks, dan antarmuka pengguna lainnya berjalan pada sisi pengguna, sementara pemrosesan data, dan klasifikasi teks dilakukan di sisi server (*back-end*). Berikut penjelasan lebih lengkap tentang *front-end* dan *back-end* :

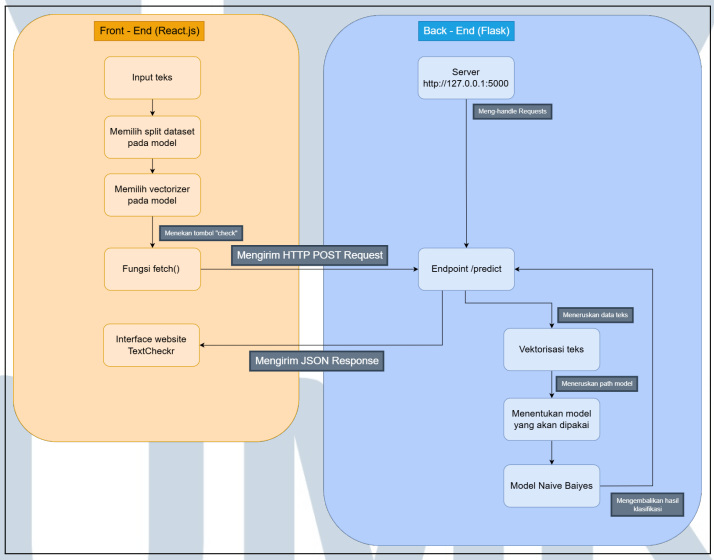
1. *Front-End*

   Front-End dikembangkan dengan *framework React.js* dan *TailwindCSS* pengguna dapat mengakses aplikasi melalui browser web yang menampilkan antarmuka interaktif untuk memasukkan teks yang akan diperiksa, *split* data pada model yang akan digunakan, dan varian *vectorizer* pada model yang akan digunakan. Setelah data itu dikirimkan, data tersebut diteruskan ke *back-end* berupa *request* dan setelah *back-end* memberi *response*, hasil dari

response akan ditampilkan dalam bentuk label klasifikasi dan persentase probabilitas untuk masing-masing kelas.

2. *Back-End*

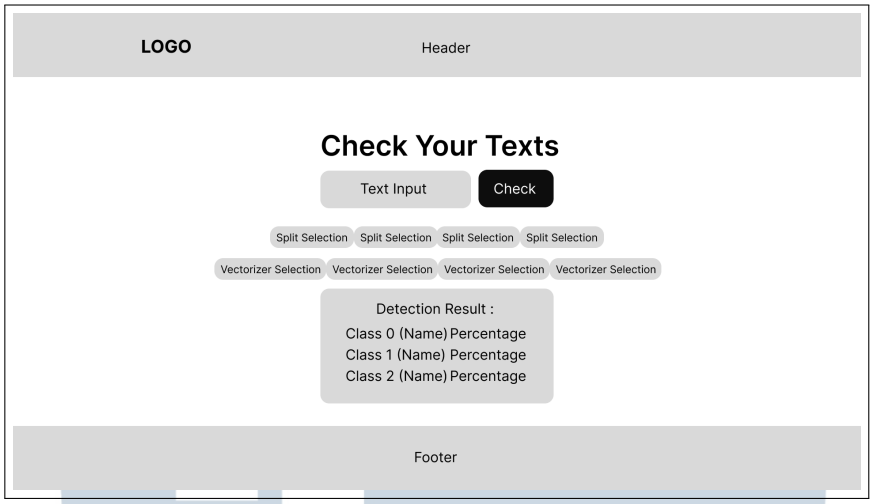
   Back-End dikembangkan menggunakan *Flask*, yaitu sebuah *framework* dari bahasa pemrograman Python yang ringan dan cocok untuk aplikasi berbasis REST API. *back-end* bertanggung jawab untuk menerima input berupa teks dan *path* model yang akan dipakai dari *front-end*, lalu memilih model dan *vectorizer* dari *path* yang diterima, kemudian melakukan vektorisasi teks dan memprosesnya menggunakan model klasifikasi teks yang telah dipilih *path*-nya, dan akhirnya mengirim hasil nya kembali ke *front-end*. Alur antara *front-end* dan *back-end* dapat dilihat di gambar 3.7.



Gambar 3.7. Alur Aplikasi *"TextCheckr"*

3.3.3 *Wireframe* Aplikasi *"TextCheckr"*

Wireframe adalah representasi visual sederhana dari antarmuka pengguna sebuah *website*, yang digunakan untuk merancang tata letak dan struktur halaman sebelum dikembangkan secara fungsional. *Wireframe* berfungsi sebagai panduan bagi *developer* dalam membangun elemen-elemen *website* seperti form input, tombol, dan area hasil, tanpa perlu fokus pada aspek estetika terlebih dahulu. Berikut tampilan *wireframe* aplikasi *"TextCheckr"* pada gambar 3.8.



Gambar 3.8. *Wireframe* Aplikasi *"TextCheckr"*

Pada aplikasi *"TextCheckr"*, *wireframe* dirancang untuk memberikan pengalaman pengguna yang intuitif dan efisien. Berikut penjelasan komponen pada *wireframe* aplikasi:

- *Header* berguna untuk menampilkan nama aplikasi dan logo aplikasi.
- *Text Input* adalah tempat pengguna dapat memasukkan teks yang ingin diperiksa. Biasanya berupa *textarea* dengan label yang jelas.
- Tombol *Check* digunakan untuk mengirim teks ke server melalui API untuk dilakukan klasifikasi.
- *Detection Result* adalah area yang menampilkan hasil klasifikasi berupa label seperti Normal, *Phishing*, atau Promo, lengkap dengan probabilitas masing-masing kelas.
- Tombol *Split Selection* digunakan untuk memilih pembagian dataset pada model yang akan dipakai.
- Tombol *Vectorizer Selection* digunakan untuk memilih *vectorizer* pada model yang akan dipakai.
- *Footer* dapat berisi informasi pengembang, hak cipta, atau tautan ke dokumentasi.