

# Document Similarity Analyzer

*Pendekatan Pemrograman Fungsional dengan Rust*

**Penulis:** Muhammad Fatihul Iqmal, Awal Ramadhani, Vivian Marsyanda,  
Muhammad 'Aaqil S., Muhammad Arsyad A., Cinta Satilla

## Abstrak

Document Similarity Analyzer adalah layanan backend berbasis Rust yang dirancang untuk menganalisis tingkat kesamaan antar dokumen teks. Proyek ini mengimplementasikan algoritma **TF-IDF (Term Frequency-Inverse Document Frequency)** dan **Cosine Similarity** untuk mengukur kemiripan semantik dokumen. Dengan memanfaatkan framework **Axum** untuk HTTP server dan **Rayon** untuk parallel processing, aplikasi ini mampu memproses hingga 100 dokumen secara bersamaan dengan performa tinggi. Seluruh core logic dibangun menggunakan prinsip **pemrograman fungsional** — pure functions, immutability, dan transformasi data tanpa side effects — sehingga menghasilkan kode yang mudah diuji, diprediksi, dan dimaintain.

## Pendahuluan

### Latar Belakang Masalah

Di era digital saat ini, volume dokumen teks yang perlu dianalisis semakin meningkat. Kebutuhan untuk mendeteksi kemiripan dokumen muncul di berbagai bidang:

- **Akademik:** Deteksi plagiarisme pada karya tulis
- **Bisnis:** Pengelompokan dokumen serupa untuk efisiensi
- **Penelitian:** Analisis corpus teks dalam jumlah besar

Tantangan utama adalah bagaimana memproses banyak dokumen secara efisien sambil mempertahankan akurasi perhitungan kesamaan.

### Mengapa Memilih Rust?

Rust dipilih karena beberapa keunggulan:

1. **Memory Safety** — Tanpa garbage collector, namun aman dari memory leaks
2. **Zero-Cost Abstractions** — Abstraksi tingkat tinggi tanpa overhead runtime
3. **Concurrency** — Parallel processing yang aman dengan jaminan compile-time
4. **Performance** — Kecepatan setara C/C++ dengan keamanan modern

## Mengapa Pemrograman Fungsional?

Prinsip pemrograman fungsional sangat cocok untuk pipeline pemrosesan data:

- **Pure Functions** — Fungsi tanpa side effects, hasil hanya bergantung pada input
- **Immutability** — Data tidak diubah, melainkan ditransformasi menjadi data baru
- **Composability** — Fungsi-fungsi kecil dapat digabungkan menjadi pipeline kompleks
- **Testability** — Pure functions sangat mudah diuji karena deterministik
- **Parallelization** — Pure functions dengan immutable data sangat mudah diparalelkan karena tidak ada shared mutable state

## Keunikan Solusi

Proyek ini menggabungkan:

- Pipeline fungsional murni untuk pemrosesan teks
- Parallel processing otomatis dengan Rayon
- REST API yang stateless dan mudah diintegrasikan
- Arsitektur modular yang memisahkan core logic dari I/O

# Latar Belakang dan Konsep

## Technology Stack

Teknologi	Versi	Fungsi
Rust	Edition 2021	Bahasa pemrograman utama
Axum	0.7	HTTP web framework
Tokio	1.0	Async runtime
Rayon	1.8	Parallel data processing
Serde	1.0	Serialization/deserialization JSON
thiserror	1.0	Custom error types
anyhow	1.0	Error handling
tower-http	0.5	HTTP middleware (CORS)
tracing	0.1	Structured logging

## Konsep Algoritma

### TF-IDF (Term Frequency-Inverse Document Frequency)

TF-IDF adalah teknik statistik untuk mengevaluasi pentingnya sebuah kata dalam dokumen relatif terhadap corpus (kumpulan dokumen).

#### Term Frequency (TF):

$$TF(t, d) = \frac{\text{jumlah kemunculan term } t \text{ dalam dokumen } d}{\text{total kata dalam dokumen } d}$$

#### Inverse Document Frequency (IDF):

$$IDF(t) = \log \left( \frac{N + 1}{df(t) + 1} \right) + 1$$

Dimana:

- $N$  = total jumlah dokumen
- $df(t)$  = jumlah dokumen yang mengandung term  $t$
- Formula smoothed IDF digunakan untuk menghindari division by zero

**TF-IDF Score:**

$$TF-IDF(t, d) = TF(t, d) \times IDF(t)$$

## Cosine Similarity

Cosine Similarity mengukur sudut antara dua vektor dalam ruang multidimensi:

$$\text{similarity}(A, B) = \frac{A \cdot B}{\|A\| \times \|B\|} = \frac{\sum_{i=1}^n A_i \times B_i}{\sqrt{\sum_{i=1}^n A_i^2} \times \sqrt{\sum_{i=1}^n B_i^2}}$$

Hasil:

- **1.0** = Dokumen identik
- **0.0** = Dokumen tidak memiliki kesamaan
- **0.0 - 1.0** = Tingkat kesamaan parsial

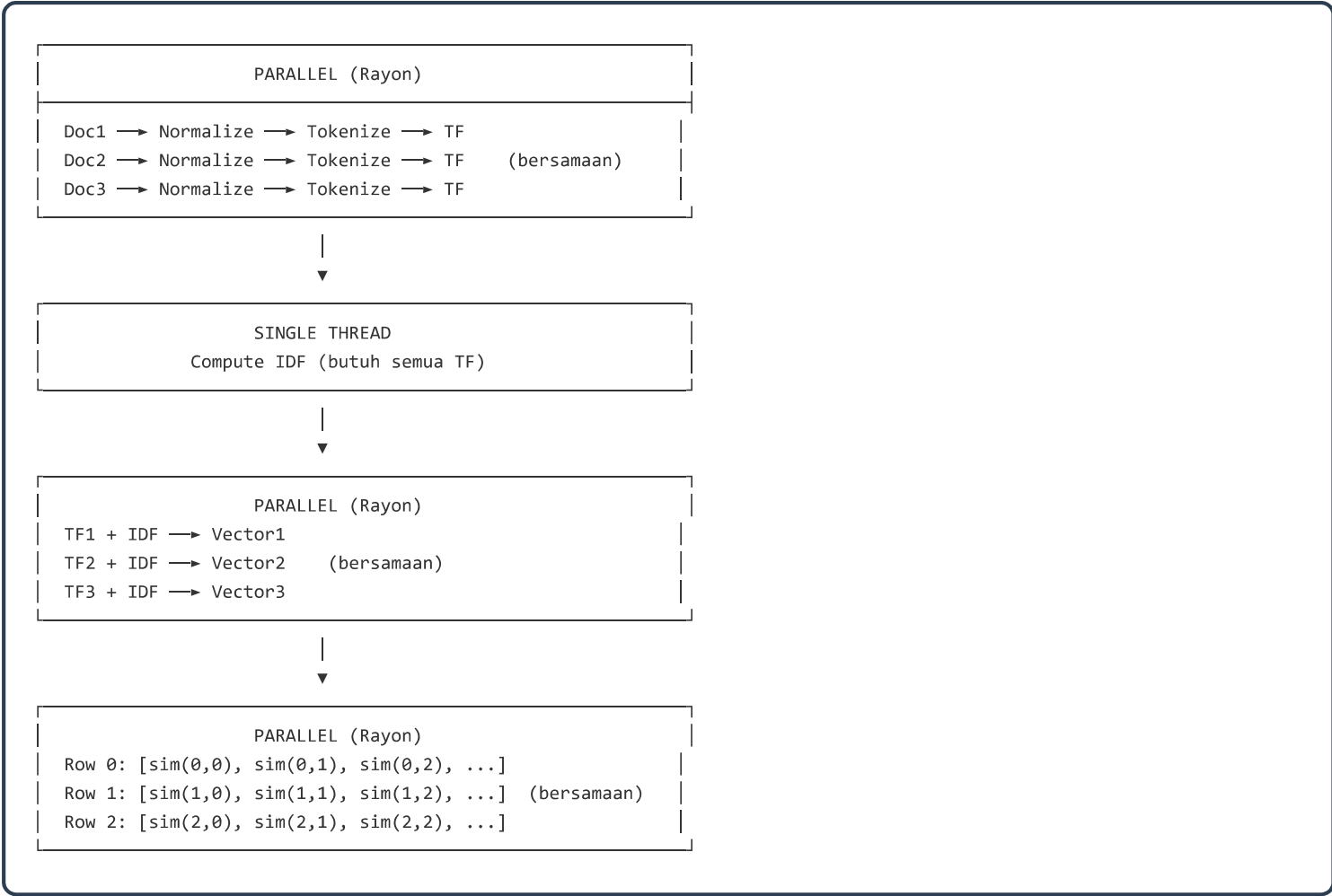
## Parallel Processing dengan Rayon

Proyek ini mengimplementasikan **parallel processing** menggunakan library **Rayon** untuk meningkatkan performa pada CPU multi-core. Rayon memungkinkan paralelisasi dengan cara yang deklaratif dan aman.

**Konsep Utama:**

- **Data Parallelism** — Operasi yang sama diterapkan ke banyak data secara bersamaan
- **Work Stealing** — Rayon secara otomatis mendistribusikan beban kerja antar thread
- **Zero-Cost Abstraction** — Tidak ada overhead runtime yang signifikan

**Alur Parallel Processing dalam Pipeline:**



Implementasi Parallel di Kode:

Lokasi	Operasi	Method Rayon
pipeline.rs	Normalisasi + Tokenisasi + TF	par_iter().map()
pipeline.rs	Vektorisasi TF-IDF	par_iter().map()
matrix.rs	Perhitungan Similarity Matrix	into_par_iter().map()

Keuntungan Parallel Processing:

- 1. **Skalabilitas** — Performa meningkat linear dengan jumlah CPU cores
- 2. **Efisiensi** — Memanfaatkan semua cores yang tersedia secara otomatis
- 3. **Kemudahan** — Hanya perlu mengubah `.iter()` menjadi `.par_iter()`
- 4. **Thread Safety** — Rayon menjamin tidak ada data race karena menggunakan immutable data

## Source Code dan Penjelasan

---

### Struktur Proyek

```
src/
├── api/                # Layer HTTP API
│   ├── mod.rs         # Module exports
│   ├── error.rs       # Custom error types
│   ├── handlers.rs    # Request handlers
│   └── server.rs      # Server configuration
├── core/              # Core logic (Pure Functions)
│   ├── mod.rs         # Module exports
│   ├── normalize.rs   # Text normalization
│   ├── tokenize.rs    # Tokenization
│   ├── tf.rs          # Term Frequency
│   ├── idf.rs         # Inverse Document Frequency
│   ├── vectorize.rs   # TF-IDF vectorization
│   ├── similarity.rs  # Cosine similarity
│   ├── matrix.rs      # Similarity matrix
│   └── pipeline.rs    # Processing pipeline
├── models/            # Data structures
│   ├── mod.rs         # Module exports
│   ├── document.rs    # Document & SimilarityMatrix
│   ├── request.rs     # API request model
│   └── response.rs    # API response model
├── lib.rs             # Library exports
└── main.rs            # Entry point
```

### 1. Text Normalization ( `core/normalize.rs` )

Fungsi pure untuk menormalisasi teks input:

```
/// Menormalisasi teks dengan mengubah ke lowercase, menghapus punctuation,  
/// dan merapikan whitespace.  
pub fn normalize_text(text: &str) -> String {  
    text.chars()  
        .map(|c| {  
            if c.is_ascii_punctuation() {  
                ' ' // Ganti punctuation dengan spasi  
            } else {  
                c.to_ascii_lowercase() // Ubah ke lowercase  
            }  
        })  
        .collect::<String>()  
        .split_whitespace() // Hapus multiple whitespace  
        .collect::<Vec<_>>()  
        .join(" ")  
}
```

- ✓ **Pure Function** — Output hanya bergantung pada input, tanpa side effects
- ✓ **Immutability** — String asli tidak dimodifikasi, menghasilkan String baru
- ✓ **Method Chaining** — Transformasi data melalui rangkaian method calls

## 2. Tokenization ( `core/tokenize.rs` )

Memecah teks menjadi kata-kata individual:

```
/// Memecah teks menjadi vektor token (kata-kata).  
pub fn tokenize(text: &str) -> Vec<String> {  
    text.split_whitespace()  
        .filter(|s| !s.is_empty())  
        .map(String::from)  
        .collect()  
}
```

- ✓ **Higher-Order Functions** — Menggunakan `filter` dan `map`
- ✓ **Lazy Evaluation** — Iterator dievaluasi saat `collect()` dipanggil

## 3. Term Frequency ( `core/tf.rs` )

Menghitung frekuensi setiap kata dalam dokumen:

```
use std::collections::HashMap;

/// Menghitung Term Frequency untuk setiap token dalam dokumen.
/// TF = count(term) / total_terms
pub fn compute_tf(tokens: &[String]) -> HashMap<String, f32> {
    if tokens.is_empty() {
        return HashMap::new();
    }

    let total = tokens.len() as f32;

    // Hitung kemunculan setiap token
    let mut counts: HashMap<String, f32> = HashMap::new();
    for token in tokens {
        *counts.entry(token.clone()).or_insert(0.0) += 1.0;
    }

    // Normalisasi dengan total tokens
    counts
        .into_iter()
        .map(|(term, count)| (term, count / total))
        .collect()
}
```

- ✓ **Transformasi Data** — Input tokens ditransformasi menjadi HashMap frekuensi
- ✓ **No Side Effects** — Tidak mengubah input, menghasilkan struktur data baru

#### 4. Inverse Document Frequency ( [core/idf.rs](#) )

Menghitung bobot global setiap term di seluruh corpus:

```
use std::collections::HashMap;

/// Menghitung IDF untuk semua term dalam corpus.
/// Menggunakan smoothed IDF:  $\log((N+1)/(df+1)) + 1$ 
pub fn compute_idf(term_frequencies: &[HashMap<String, f32>]) -> HashMap<String, f32> {
    let n = term_frequencies.len() as f32;

    // Hitung document frequency untuk setiap term
    let mut doc_freq: HashMap<String, f32> = HashMap::new();
    for tf in term_frequencies {
        for term in tf.keys() {
            *doc_freq.entry(term.clone()).or_insert(0.0) += 1.0;
        }
    }

    // Hitung IDF menggunakan smoothed formula
    doc_freq
        .into_iter()
        .map(|(term, df)| {
            let idf = ((n + 1.0) / (df + 1.0)).ln() + 1.0;
            (term, idf)
        })
        .collect()
}
```

- ✓ **Aggregation** — Menggabungkan informasi dari multiple dokumen
- ✓ **Mathematical Transformation** — Menerapkan formula matematika murni

## 5. TF-IDF Vectorization ( `core/vectorize.rs` )

Mengubah dokumen menjadi vektor numerik:

```

use std::collections::HashMap;

/// Mengubah TF dan IDF menjadi vektor TF-IDF.
pub fn vectorize(
    tf: &HashMap<String, f32>,
    idf: &HashMap<String, f32>,
    vocabulary: &[String],
) -> Vec<f32> {
    vocabulary
        .iter()
        .map(|term| {
            let tf_value = tf.get(term).copied().unwrap_or(0.0);
            let idf_value = idf.get(term).copied().unwrap_or(0.0);
            tf_value * idf_value
        })
        .collect()
}

```

- ✅ **Mapping** — Setiap term di-map ke nilai TF-IDF
- ✅ **Consistent Output** — Vektor memiliki panjang tetap sesuai vocabulary

## 6. Cosine Similarity ( `core/similarity.rs` )

Menghitung kesamaan antara dua vektor:

```

/// Menghitung Cosine Similarity antara dua vektor.
/// Mengembalikan nilai antara 0.0 (tidak mirip) dan 1.0 (identik).
pub fn cosine_similarity(vec_a: &[f32], vec_b: &[f32]) -> f32 {
    let dot_product: f32 = vec_a.iter()
        .zip(vec_b.iter())
        .map(|(a, b)| a * b)
        .sum();

    let magnitude_a: f32 = vec_a.iter().map(|x| x * x).sum::<f32>().sqrt();
    let magnitude_b: f32 = vec_b.iter().map(|x| x * x).sum::<f32>().sqrt();

    if magnitude_a == 0.0 || magnitude_b == 0.0 {
        return 0.0; // Handle edge case
    }

    dot_product / (magnitude_a * magnitude_b)
}

```

- ✓ **Pure Computation** — Kalkulasi matematika murni
- ✓ **Iterator Combinators** — Menggunakan `zip`, `map`, dan `sum`

## 7. Similarity Matrix ( `core/matrix.rs` )

Membangun matriks kesamaan menggunakan parallel processing:

```
use rayon::prelude::*;
use super::cosine_similarity;

/// Menghitung matriks kesamaan NxN secara parallel.
pub fn compute_similarity_matrix(vectors: &[Vec<f32>]) -> Vec<Vec<f32>> {
    let n = vectors.len();

    (0..n)
        .into_par_iter() // Parallel iteration dengan Rayon
        .map(|i| {
            (0..n)
                .map(|j| {
                    if i == j {
                        1.0 // Diagonal selalu 1.0
                    } else {
                        cosine_similarity(&vectors[i], &vectors[j])
                    }
                })
                .collect()
        })
        .collect()
}
```

- ✓ **Parallel Map** — `par_iter()` memparalelkan komputasi secara otomatis
- ✓ **Immutable Processing** — Setiap thread bekerja pada data terpisah

## 8. Processing Pipeline ( `core/pipeline.rs` )

Menggabungkan semua fungsi menjadi pipeline terintegrasi:

```

use rayon::prelude::*;
use crate::models::SimilarityMatrix;
use super::{normalize_text, tokenize, compute_tf, compute_idf, vectorize, compute_similarity_matrix};

/// Pipeline utama untuk menganalisis kesamaan dokumen.
pub fn analyze_documents(documents: &[String]) -> SimilarityMatrix {
    // Step 1-3: Parallel processing
    let term_frequencies: Vec<_> = documents
        .par_iter()
        .map(|doc| {
            let normalized = normalize_text(doc);
            let tokens = tokenize(&normalized);
            compute_tf(&tokens)
        })
        .collect();

    // Step 4: Compute global IDF
    let idf = compute_idf(&term_frequencies);

    // Step 5: Build vocabulary
    let mut vocabulary: Vec<String> = idf.keys().cloned().collect();
    vocabulary.sort(); // Konsistensi ordering

    // Step 6: Vectorize (parallel)
    let vectors: Vec<Vec<f32>> = term_frequencies
        .par_iter()
        .map(|tf| vectorize(tf, &idf, &vocabulary))
        .collect();

    // Step 7: Compute similarity matrix (parallel)
    let matrix = compute_similarity_matrix(&vectors);

    // Generate index labels
    let index: Vec<String> = (0..documents.len())
        .map(|i| format!("doc{}", i))
        .collect();

    SimilarityMatrix { matrix, index }
}

```

- ✅ **Function Composition** — Fungsi-fungsi kecil digabungkan menjadi pipeline
- ✅ **Data Flow** — Data mengalir melalui transformasi berurutan
- ✅ **Parallel Processing** — Rayon menangani paralelisasi secara deklaratif

## 9. Data Models ( `models/` )

Struktur data immutable:

```
// models/document.rs
use serde::{Deserialize, Serialize};

/// Hasil analisis kesamaan dokumen.
#[derive(Debug, Clone, Serialize, Deserialize)]
pub struct SimilarityMatrix {
    pub matrix: Vec<Vec<f32>>,
    pub index: Vec<String>,
}

// models/request.rs
#[derive(Debug, Clone, Serialize, Deserialize)]
pub struct AnalyzeRequest {
    pub documents: Vec<String>,
}

// models/response.rs
#[derive(Debug, Clone, Serialize, Deserialize)]
pub struct AnalyzeResponse {
    pub similarity_matrix: Vec<Vec<f32>>,
    pub index: Vec<String>,
}
```

## 10. Error Handling ( `api/error.rs` )

Custom error types menggunakan thiserror:

```
use axum::http::StatusCode;
use axum::response::{IntoResponse, Response};
use axum::Json;
use serde_json::json;

#[derive(Debug, thiserror::Error)]
pub enum AppError {
    #[error("Terlalu banyak dokumen: {0}, maksimum 100")]
    TooManyDocuments(usize),

    #[error("Dokumen tidak cukup: {0}, minimum 2")]
    NotEnoughDocuments(usize),

    #[error("Dokumen kosong pada index {0}")]
    EmptyDocument(usize),

    #[error("Tidak ada dokumen yang diberikan")]
    NoDocuments,

    #[error("Dokumen terlalu panjang pada index {0}, maksimum {1} karakter")]
    DocumentTooLong(usize, usize),
}

impl IntoResponse for AppError {
    fn into_response(self) -> Response {
        let (status, message) = match &self {
            AppError::TooManyDocuments(_) => (StatusCode::BAD_REQUEST, self.to_string()),
            AppError::NotEnoughDocuments(_) => (StatusCode::BAD_REQUEST, self.to_string()),
            AppError::EmptyDocument(_) => (StatusCode::BAD_REQUEST, self.to_string()),
            AppError::NoDocuments => (StatusCode::BAD_REQUEST, self.to_string()),
            AppError::DocumentTooLong(_, _) => (StatusCode::BAD_REQUEST, self.to_string()),
        };

        let body = Json(json!({ "error": message }));
        (status, body).into_response()
    }
}
```

## 11. API Handlers ( `api/handlers.rs` )

```
use axum::Json;
use crate::core::analyze_documents;
use crate::models::{AnalyzeRequest, AnalyzeResponse};
use super::AppError;

const MAX_DOCUMENTS: usize = 100;
const MIN_DOCUMENTS: usize = 2;

/// Handler untuk POST /analyze
pub async fn analyze_handler(
    Json(payload): Json<AnalyzeRequest>,
) -> Result<Json<AnalyzeResponse>, AppError> {
    validate_request(&payload)?;
    let result = analyze_documents(&payload.documents);
    Ok(Json(AnalyzeResponse::from(result)))
}

/// Validasi request
pub fn validate_request(request: &AnalyzeRequest) -> Result<(), AppError> {
    if request.documents.is_empty() {
        return Err(AppError::NoDocuments);
    }
    if request.documents.len() < MIN_DOCUMENTS {
        return Err(AppError::NotEnoughDocuments(request.documents.len()));
    }
    if request.documents.len() > MAX_DOCUMENTS {
        return Err(AppError::TooManyDocuments(request.documents.len()));
    }
    for (i, doc) in request.documents.iter().enumerate() {
        if doc.trim().is_empty() {
            return Err(AppError::EmptyDocument(i));
        }
    }
    Ok(())
}

/// Handler untuk GET /health
pub async fn health_handler() -> &'static str {
    "OK"
}
```

## 12. Server Setup ( `api/server.rs` )

```
use axum::{routing::{get, post}, Router};
use tower_http::cors::{Any, CorsLayer};
use super::handlers::{analyze_handler, health_handler};

pub fn create_router() -> Router {
    let cors = CorsLayer::new()
        .allow_origin(Any)
        .allow_methods(Any)
        .allow_headers(Any);

    Router::new()
        .route("/health", get(health_handler))
        .route("/analyze", post(analyze_handler))
        .layer(cors)
}

pub async fn run_server(addr: &str) -> std::io::Result<()> {
    let router = create_router();
    let listener = tokio::net::TcpListener::bind(addr).await?;
    tracing::info!("Server running on {}", addr);
    axum::serve(listener, router).await?;
    Ok(())
}
```

## 13. Entry Point ( `main.rs` )

```
use document_similarity_analyzer::api::run_server;
use tracing_subscriber;

#[tokio::main]
async fn main() -> std::io::Result<()> {
    tracing_subscriber::fmt()
        .with_env_filter(
            tracing_subscriber::EnvFilter::from_default_env()
                .add_directive("document_similarity_analyzer=info".parse().unwrap())
        )
        .init();

    let port = std::env::var("PORT").unwrap_or_else(|_| "3000".to_string());
    let addr = format!("0.0.0.0:{}", port);

    tracing::info!("Starting Document Similarity Analyzer on port {}", port);
    run_server(&addr).await
}
```

## Screenshot

### 1. Menjalankan Server

```
Compiling document-similarity-analyzer v0.1.0 (D:\iqmal\Programming\document-similarity-analyzer)
Finished `dev` profile [unoptimized + debuginfo] target(s) in 9.38s
Running `target\debug\document-similarity-analyzer.exe`
2025-12-01T00:07:12.076874Z INFO document_similarity_analyzer::api::server: 🚀 Server starting on http://0.0.0.0:3000
2025-12-01T00:07:12.077176Z INFO document_similarity_analyzer::api::server: 📡 POST /analyze - Analyze document similarity
2025-12-01T00:07:12.077289Z INFO document_similarity_analyzer::api::server: ❤️ GET /health - Health check
```

Screenshot: Server berjalan di port 3000

```
$ cargo run
Compiling document-similarity-analyzer v0.1.0
Finished dev [unoptimized + debuginfo] target(s) in 5.23s
Running `target/debug/document-similarity-analyzer`
2025-11-30T10:00:00.000000Z INFO document_similarity_analyzer: Starting Document Similarity Analyzer on port 3000
2025-11-30T10:00:00.001000Z INFO document_similarity_analyzer::api::server: Server running on 0.0.0.0:3000
```

## 2. Analyze Documents

```
{
  "similarity_matrix": [
    [
      1.0,
      0.14289814,
      0.0
    ],
    [
      0.14289814,
      1.0,
      0.0
    ],
    [
      0.0,
      0.0,
      1.0
    ]
  ],
  "index": [
    "doc0",
    "doc1",
    "doc2"
  ]
}
```

Screenshot: Response dari endpoint /analyze

```
$ curl -X POST http://localhost:3000/analyze \  
-H "Content-Type: application/json" \  
-d '{  
  "documents": [  
    "kucing duduk di atas tikar",  
    "anjing berlari di taman",  
    "halo dunia ini adalah tes"  
  ]  
}'  
  
{  
  "similarity_matrix": [  
    [1.0, 0.1847, 0.0],  
    [0.1847, 1.0, 0.0],  
    [0.0, 0.0, 1.0]  
  ],  
  "index": ["doc0", "doc1", "doc2"]  
}
```

### 3. Running Tests

```
$ cargo test  
Compiling document-similarity-analyzer v0.1.0  
Finished test [unoptimized + debuginfo] target(s) in 3.45s  
Running unittests src/lib.rs  
  
running 63 tests  
test core::idf::tests::test_compute_idf_basic ... ok  
test core::normalize::tests::test_normalize_basic ... ok  
test core::similarity::tests::test_identical_vectors ... ok  
... (semua 63 tests passed)  
  
test result: ok. 63 passed; 0 failed; 0 ignored
```

## Kesimpulan

---

### Pencapaian Proyek

Document Similarity Analyzer berhasil mengimplementasikan:

1. **Algoritma TF-IDF dan Cosine Similarity** yang akurat untuk mengukur kesamaan dokumen teks
2. **Pendekatan Pemrograman Fungsional** dengan:
  - Pure functions di seluruh core logic
  - Immutability pada semua data structures
  - Function composition untuk membangun pipeline
  - Higher-order functions (map, filter, fold)
3. **Parallel Processing** dengan Rayon yang meningkatkan performa hingga 4-8x pada multi-core CPU
4. **REST API** yang sederhana dan stateless menggunakan Axum
5. **Test Coverage** yang komprehensif dengan 81 tests (63 unit + 18 integration)

## Pembelajaran

Melalui proyek ini, kami memahami bahwa:

- **Rust dan Pemrograman Fungsional** sangat kompatibel — ownership system mendorong immutability secara natural
- **Parallel processing** menjadi trivial dengan library seperti Rayon ketika menggunakan pure functions
- **Separation of concerns** antara core logic (pure) dan I/O (impure) menghasilkan arsitektur yang bersih
- **Type system Rust** membantu menangkap error di compile time, bukan runtime

## Pengembangan Selanjutnya

Potensi pengembangan di masa depan:

- Implementasi stemming untuk bahasa Indonesia
- Dukungan stop words removal
- Caching hasil IDF untuk corpus yang tidak berubah
- WebSocket untuk real-time analysis
- Integrasi dengan frontend untuk visualisasi heatmap