Dokumentasi API (RESTful)

Herta’s Vibe Checker

+ Dokumen Implementasi & Integrasi Lanjurtan

PENDAHULUAN

Dokumentasi ini menjelaskan penggunaan **Herta’s Vibe Checker API** yang memungkinkan integrator atau developer untuk memanfaatkan layanan analisis vibe melalui protokol HTTP. Semua endpoint mendukung format **REST** dengan request dan response menggunakan JSON. Untuk keamanan, API ini menerapkan autentikasi **OAuth 2.0** berbasis token, sehingga setiap request (kecuali yang dikecualikan) harus menyertakan token akses yang valid di header. Seluruh komunikasi harus melalui **HTTPS** (port 443) demi menjaga kerahasiaan data.

Basis URL layanan (misal untuk produksi) akan diinformasikan. Setiap integrator harus terlebih dahulu mendaftarkan aplikasi untuk mendapatkan `client_id` dan `client_secret` unik yang digunakan dalam proses OAuth2. Berikut detail tiap aspek: autentikasi, endpoint-endpoint GET/POST, format pesan, dan contoh penggunaan.

Rhio Bimo | 13523123

“The good stuff is further ahead. Don't stop now.”

-Herta

Autentikasi (OAuth 2.0)

Herta's Vibe Checker API menggunakan kerangka **OAuth 2.0** sebagai mekanisme otorisasi. Sebelum memanggil endpoint layanan, client harus memperoleh **access token** melalui proses OAuth2 **Client Credentials Grant** dengan sedikit modifikasi penambahan "math challenge". Konsepnya serupa dengan standar OAuth2 pada umumnya (seperti implementasi BCA API) di mana `client_id` dan `client_secret` digunakan untuk mendapatkan token. Bedanya, di sini server memberikan tantangan matematika sederhana yang harus dijawab oleh client saat meminta token sebagai lapisan keamanan tambahan.

1. Meminta Math Challenge
`GET /math_challenge`
- **Deskripsi:** Endpoint ini mengembalikan sebuah tantangan matematika sederhana (misal penjumlahan acak) dan sebuah `challenge_id` yang diperlukan untuk fase berikutnya. Tantangan ini berlaku sekali pakai dan memiliki masa berlaku pendek (misal 5 menit).
- **Otentikasi:** Tidak memerlukan token (public), namun dibatasi dari sisi rate-limit. Bisa diakses sebelum mendapatkan token.
- **Request:** Tidak ada parameter khusus.
- **Response:** JSON dengan format:

```
{
  "challenge_id": "XYZ12345",
  "question": "12+5=?"
}
```

Contoh di atas berarti server mengirim ID unik `XYZ12345` dan pertanyaan penjumlahan "12+5=?". Client harus menghitung jawabannya (17) untuk digunakan pada request token.

2. Mendapatkan Access Token
`POST /oauth/token`
- **Deskripsi:** Endpoint untuk memperoleh access token OAuth2 menggunakan grant type *client_credentials* (aplikasi-ke-aplikasi) dengan menjawab tantangan yang diberikan.
- **Otentikasi:** Basic Auth dengan header `Authorization: Basic <base64_encode(client_id:client_secret)>` diperlukan. Ini memastikan hanya client terdaftar yang bisa meminta token.
- **Headers:**
- `Authorization: Basic <encoded_credentials>`
- `Content-Type: application/x-www-form-urlencoded`
- **Body:** Form URL-encoded dengan field berikut:
- `grant_type`: harus diisi `"client_credentials"`
- `challenge_id`: ID tantangan yang diperoleh dari langkah sebelumnya (`challenge_id` dari `/math_challenge`).
- `challenge_answer`: jawaban tantangan matematika yang diberikan.
- **Response:** Jika sukses, mengembalikan HTTP 200 dengan body JSON:

```
{
  "access_token": "<token-string>",
  "token_type": "Bearer",
  "expires_in": 3600
}
```

Field `access_token` adalah string token (misal format JWT atau opaque string) yang harus digunakan pada request API lainnya (pada header Authorization sebagai Bearer). `expires_in` menyatakan masa berlaku token dalam detik (contoh di atas 3600 detik = 1 jam).

- **Contoh Request Token (Pseudo):**

  ```
  POST /oauth/token HTTP/1.1
  Host: api.herta-vibe.com
  Authorization: Basic SGlndXNlckFwcElEOmFhYmJjYzEyMzQ=   <-- base64(clie
  nt_id:client_secret)
  Content-Type: application/x-www-form-urlencoded

  grant_type=client_credentials&challenge_id=XYZ12345&challenge_answer=17
  ```

  Contoh Response:

  ```
  HTTP/1.1 200 OK
  {
    "access_token": "eyJhbGciOiJI... (JWT token) ...",
    "token_type": "Bearer",
    "expires_in": 3600,
    "scope": "vibe.read vibe.write"
  }
  ```

  Jika `client_id/secret` salah atau tantangan salah, response bisa berupa error 401 Unauthorized dengan body berisi error message.

Penggunaan Token: Setelah memperoleh `access_token`, setiap request ke endpoint layanan (selain `/oauth/token` dan `/math_challenge`) harus mencantumkan header:

```
Authorization: Bearer <access_token>
```

Token ini akan diverifikasi pada server setiap kali (bisa melalui validasi signature JWT dan cek masa berlaku, atau pencocokan dengan data server). Jika token expired atau invalid, server akan merespon 401/403. Pastikan untuk refresh token sebelum kadaluarsa (dengan memanggil `/oauth/token` lagi apabila perlu, atau jika menggunakan JWT, generate ulang setelah expired).

Daftar Endpoint Layanan

Berikut adalah daftar endpoint utama yang tersedia dalam Herta's Vibe Checker API beserta detil penggunaannya:

1. Health Check

Endpoint: `GET /health`
- **Deskripsi:** Memeriksa status kesehatan layanan. Berguna untuk monitoring.
- **Otentikasi:** *Optional/Public*. (Endpoint ini dapat dibuat public tanpa token untuk kemudahan pengecekan oleh load balancer atau uptime monitor. Tidak mengandung data sensitif.)
- **Request:** Tidak ada parameter.
- **Response:** Jika layanan aktif, merespon dengan kode 200 dan body minimal:

```
{ "status": "OK", "version": "1.0.0" }
```

Misalnya field `version` menunjukkan versi current deployment aplikasi.
- **Error:** Jika layanan tidak sehat (misal dependensi down), bisa merespon 500 dengan `status: "error"` atau serupa, namun biasanya jika API server sendiri down tidak ada respon sama sekali.

2. Get Service Stats

Endpoint: `GET /stats`
- **Deskripsi:** Mengambil statistik umum layanan. Misalnya jumlah total teks yang telah dianalisis, breakdown persentase sentimen, rata-rata waktu respon, dsb. Dapat juga memberikan statistik penggunaan khusus client (misal jumlah request yang telah dilakukan oleh client tersebut bulan ini) jika diperlukan.
- **Otentikasi: Wajib** (Bearer token harus disertakan).
- **Request:** Dapat menyediakan parameter query opsional. Contoh: `?scope=client` untuk stats khusus client pemanggil, atau `?scope=global` untuk stats global. Jika tidak dispesifikkan, default mungkin mengembalikan global non-sensitif.
- **Response:** HTTP 200 + JSON. Contoh format:

```
{
  "total_texts_analyzed": 125430,
  "positive_percentage": 0.62,
  "neutral_percentage": 0.18,
  "negative_percentage": 0.20,
  "avg_response_time_ms": 120,
  "uptime": "99.98%"
}
```

Penjelasan contoh: Layanan telah menganalisis 125 ribu teks, dengan 62% berisi vibe positif, 20% negatif, sisanya netral; waktu respon rata-rata ~120 ms; uptime 99.98%. Untuk scope client, misal bisa ditambahkan field `texts_analyzed_by_client`.
- **Error:** 401 Unauthorized jika token tidak valid/absent.

3. Get Math Challenge

(Sudah dibahas di bagian Autentikasi, disertakan di sini untuk kelengkapan.)
Endpoint: `GET /math_challenge`
- Memberikan tantangan matematika + ID, sebagai tahap awal memperoleh token.
- Otentikasi: tidak perlu token, tapi gunakan Basic Auth atau mekanisme antispam lain jika diperlukan. *(Atau bisa juga dibuat public dengan rate-limit ketat per IP.)*
- Response: `{ "challenge_id": "...", "question": "..." }` seperti telah dicontohkan.

4. Get Previous Vibes

Endpoint: `GET /vibes`
- **Deskripsi:** Mengambil hasil-hasil analisis terdahulu (riwayat) milik client yang memanggil. Endpoint ini memudahkan client mendapatkan data vibe lama tanpa harus menyimpan sendiri di sisi mereka.
- **Otentikasi: Wajib** (Bearer token). Data yang dikembalikan otomatis terfilter hanya milik token/client tersebut.
- **Request:** Mungkin mendukung parameter query untuk paging atau filter: - `?limit=100` (jumlah maksimum entri, default misal 50), - `?offset=50` (untuk pagination), - atau `?since=<timestamp>` (jika ingin ambil setelah waktu tertentu).
Jika tanpa parameter, misal default mengembalikan 50 entri terbaru.
- **Response:** HTTP 200 + JSON array of results. Contoh:

```
{
  "history": [
    {
      "id": "res_9876543210",
      "text": "I love this game!",
      "vibe": "positive",
      "score": 0.95,
      "timestamp": "2025-08-10T06:12:30Z"
    },
    {
      "id": "res_9876543209",
      "text": "This is so frustrating... >:(",
      "vibe": "negative",
      "score": 0.89,
      "timestamp": "2025-08-10T05:57:02Z"
    },
    ...
```

```
  ]
}
```

Field `id` adalah ID unik hasil (bisa berupa kombinasi atau nomor incremental), `text` adalah teks asli (opsional dikembalikan, tergantung kebijakan penyimpanan), `vibe` adalah label sentimen, `score` kepercayaan, dan `timestamp` waktu analisis.
- **Error:** 401 jika token invalid. 400 jika parameter query tidak valid.

5. Check Vibe (Single Text)

Endpoint: `POST /vibe-check/single` *(alternatif: bisa saja cukup `/vibe-check` tanpa "single", di sini dibuat eksplisit)*
- **Deskripsi:** Menganalisis vibe untuk satu input teks. Ini endpoint utama untuk case per teks.
- **Otentikasi: Wajib** (Bearer token).
- **Request:** - **Method & URL:** `POST /vibe-check/single`
- **Headers:** `Content-Type: application/json`, plus `Authorization: Bearer <token>`.
- **Body:** JSON dengan format, misal:

```
{ "text": "contoh kalimat yang akan dianalisis" }
```

Panjang maksimal teks bisa dibatasi (misal 500 karakter) tergantung kebijakan. Jika melebihi, server dapat merespon error 413 (Payload Too Large) atau memotong teks (namun lebih baik client yang memastikan). Encoding teks dalam UTF-8.
- **Response:** Jika sukses, HTTP 200 dengan body JSON hasil analisis:

```
{
  "id": "res_1234567890",
  "text": "contoh kalimat yang akan dianalisis",
  "vibe": "neutral",
  "score": 0.45,
  "detail": {
    "positive_score": 0.2,
    "negative_score": 0.1,
    "neutral_score": 0.7
  }
}
```

Penjelasan: `id` adalah ID unik untuk hasil analisis ini (bisa digunakan untuk referensi di /vibes nanti), `text` mengulangi teks input (atau bisa tidak dikembalikan demi efisiensi, tergantung kebutuhan), `vibe` adalah label sentimen dominan, `score` adalah confidence terkait label tersebut (0 sampai 1). Bagian `detail` opsional menunjukkan skor masing-masing kategori (jika metode analisis mendukung, di sini misal neutral 0.7 yang tertinggi sehingga vibe "neutral").

- **Error Handling:**
- 400 Bad Request jika input tidak valid (misal field `text` kosong atau bukan string). Response bisa berupa: `{ "error": "Invalid input, 'text' is required" }`.
- 401 Unauthorized jika token salah.
- 500 Internal Server Error jika terjadi kegagalan di server (misal modul AI error).

- **Contoh Request/Response:**

```
POST /vibe-check/single
Authorization: Bearer eyJhbGciOi...  (token)
Content-Type: application/json

{ "text": "I am extremely happy with the service!" }
```

Response: *(misal)*

```
{
  "id": "res_0011223344",
  "vibe": "positive",
  "score": 0.98,
  "detail": {
    "positive_score": 0.98,
    "negative_score": 0.01,
    "neutral_score": 0.01
  }
}
```

6. Check Vibe (Multiple Texts)

Endpoint: `POST /vibe-check/batch` *(atau `/vibe-check/multiple` sesuai preferensi naming)*
- **Deskripsi:** Menganalisis vibe untuk beberapa teks dalam satu panggilan. Digunakan untuk efisiensi apabila aplikasi klien sudah memiliki sekumpulan teks (misal 10 baris chat) dan ingin dikirim sekaligus.
- **Otentikasi: Wajib** (Bearer token).
- **Request:**
- **Method & URL:** `POST /vibe-check/batch`
- **Headers:** `Content-Type: application/json`, plus Authorization token.
- **Body:** JSON, misalnya:

```
{ "texts": [
    "teks pertama untuk dicek",
    "teks kedua...",
    "dst."
  ]
}
```

Yaitu objek dengan array `texts`. Limit jumlah teks per request dapat ditentukan (misal maksimal 100 teks per batch untuk mencegah beban berlebih).
- **Response:** Jika sukses, HTTP 200 dengan format JSON:

```
{
  "results": [
    {
      "id": "res_001",
      "text": "teks pertama untuk dicek",
      "vibe": "neutral",
      "score": 0.50
    },
    {
      "id": "res_002",
      "text": "teks kedua...",
      "vibe": "negative",
      "score": 0.80
    },
    ...
  ]
}
```

Merupakan array `results` yang masing-masing elemen strukturnya sama seperti output single text (tanpa field detail untuk ringkas). Urutan output dijamin sama dengan urutan input.
- **Error Handling:**
- 400 Bad Request jika format input salah (misal tidak ada field `texts` atau bukan array of string).
- 413 Payload Too Large jika jumlah teks terlalu banyak dalam satu batch (sesuai limit yang ditentukan).
- Error lain mirip dengan endpoint single.

- **Contoh:**

  ```
  POST /vibe-check/batch
  Authorization: Bearer <token>
  Content-Type: application/json

  {
    "texts": [
      "I love this stream!",
      "This game is terrible :("
    ]
  }
  ```

  Response:

```
{
  "results": [
    {
      "id": "res_batch001_item1",
      "vibe": "positive",
      "score": 0.85
    },
    {
      "id": "res_batch001_item2",
      "vibe": "negative",
      "score": 0.90
    }
  ]
}
```

7. (Opsional) Endpoint Lain

Tergantung kebutuhan, API dapat dikembangkan dengan endpoint tambahan, misalnya:
- `DELETE /vibes/{id}` untuk menghapus hasil tertentu (misal jika ada permintaan hapus data).
- `GET /vibes/{id}` untuk mengambil detail satu hasil tertentu.
- `POST /feedback` untuk memungkinkan pengguna memberikan umpan balik apakah label otomatis akurat atau tidak (ini bisa membantu retraining model).

Saat ini fokus endpoint sesuai scope yang telah dijelaskan.

Contoh Alur Penggunaan API

Untuk menggambarkan bagaimana komponen-komponen API digunakan bersama, berikut adalah contoh alur untuk sebuah aplikasi client menggunakan Herta's Vibe Checker:

1. **Autentikasi:** Aplikasi client terlebih dahulu melakukan `GET /math_challenge`, lalu `POST /oauth/token` dengan jawaban tantangan dan kredensial untuk mendapatkan `access_token`. Token ini disimpan di sisi client (misal in-memory atau cache) dan akan digunakan dalam header permintaan selanjutnya.

2. **Analisis Single:** Ketika ada event (misal user interface menekan tombol "Check Vibe" atas suatu komentar), aplikasi client memanggil `POST /vibe-check/single` membawa token di header dan teks di body. API mengembalikan hasil vibe yang kemudian ditampilkan ke pengguna (misal "Vibe: Positive 👍").

3. **Analisis Batch:** Aplikasi client (misal bot moderator) secara berkala mengumpulkan 10 komentar terbaru dari chat, lalu mengirim dalam satu batch melalui `POST /vibe-check/batch`. Hasil yang didapat misalnya menunjukkan 2 negatif, 8 positif – bot tersebut lalu dapat mengirim peringatan jika ada terlalu banyak vibe negatif atau menyoroti salah satu komen negatif.

4. **Melihat Riwayat:** Pengguna di dashboard ingin melihat rekap vibe hari ini, maka front-end memanggil `GET /vibes?limit=100` untuk mendapatkan 100 hasil terbaru. Data ini kemudian ditampilkan dalam bentuk tabel atau grafik (misal grafik pie positif/negatif).

5. **Statistik:** Untuk audit bulanan, sistem menarik data `GET /stats?scope=client` untuk melihat berapa banyak pemakaian API bulan ini dan memastikan tidak melebihi kuota.

Setiap langkah di atas disertai verifikasi token oleh server. Jika token kadaluarsa, client akan mengulangi langkah autentikasi untuk refresh token.

Keamanan API

Seperti disinggung, API ini menerapkan praktik keamanan standar:
- **OAuth2.0** dengan **Client Credentials Grant** memastikan hanya aplikasi terdaftar dapat akses. (Jika diperlukan akses atas nama user end, bisa ditambah OAuth2 Authorization Code flow, tapi untuk scope layanan ini tidak diperlukan).
- Komunikasi wajib **HTTPS** (TLS).
- **Rate Limiting:** Server menerapkan rate limit per IP atau per `client_id` untuk mencegah abuse (misal max 100 requests per menit untuk endpoint vibe-check, dsb).
- **Input Validation:** Semua input dari client divalidasi untuk mencegah serangan injeksi atau payload berbahaya. Hanya teks yang diizinkan, dengan panjang wajar.
- **Logging & Monitoring:** Sistem mencatat log akses (audit log) termasuk waktu, `client_id`, endpoint, dan hasil (sukses/gagal) untuk keperluan monitoring keamanan.
- **Error Details:** Server tidak mengungkap detail internal berlebihan saat error (untuk menghindari bocoran informasi yang bisa dimanfaatkan attacker). Hanya pesan secukupnya (misal “Internal Server Error” tanpa stack trace).

Dengan dokumen di atas, diharapkan developer dapat memahami cara menggunakan API Herta’s Vibe Checker. Selalu uji dahulu di lingkungan sandbox (jika disediakan) sebelum integrasi ke produksi. Apabila ada perubahan versi API (misal v2 dengan pola endpoint berbeda), dokumentasi akan disediakan terpisah.

Dokumen Implementasi & Integrasi Lanjutan

Teknologi Implementasi

Layanan Herta's Vibe Checker diimplementasikan dengan pilihan teknologi modern yang mendukung kebutuhan NLP dan skalabilitas:

- **Bahasa Pemrograman & Framework:** Implementasi backend ditulis dalam **Python** (sesuai preferensi untuk NLP) menggunakan framework web **FastAPI** (atau Flask) untuk membangun REST API. FastAPI dipilih karena kinerja yang baik, dukungan asynchronous (penting untuk menangani banyak request I/O-bound secara efisien), serta dokumentasi OpenAPI otomatis yang dihasilkan. Alternatif lain yang juga disiapkan meliputi **Node.js** (dengan Express/Koa) atau **Go** (dengan Gin/Fiber) jika diperlukan, namun Python menjadi utama demi kemudahan integrasi dengan pustaka AI.

- **NLP & AI Engine:** Komponen analisis vibe memanfaatkan model **Machine Learning NLP**. Pada versi awal, digunakan model sentiment analysis pre-trained (misal **NLTK/VADER** untuk sesuatu yang ringan atau **HuggingFace Transformers** seperti model BERT fine-tuned untuk sentiment). Model ini diakses via Python (bisa dimuat di memory dan dipanggil tiap request). Untuk teks multi-bahasa, dapat ditambahkan model per bahasa atau model multilingual. Selain itu, ada pipeline preprocessing (tokenization, cleaning) sebelum inferensi model untuk akurasi lebih baik. *Contoh:* menggunakan pustaka `transformers` untuk memuat model `"nlptown/bert-base-multilingual-uncased-sentiment"` yang dapat mendeteksi sentiment berbagai bahasa.

- **Database:** Untuk penyimpanan data, digunakan **PostgreSQL** (relational database) yang menyimpan tabel-tabel seperti yang dijelaskan di ERD. PostgreSQL dipilih karena reliabilitas dan dukungan transaksi (konsistensi data). Alternatifnya, *NoSQL* **MongoDB** dapat dipakai jika data hasil analisis ingin disimpan sebagai dokumen JSON fleksibel, tapi relational sudah mencukupi. Demi kinerja pada operasi baca intensif (seperti query riwayat), index ditambahkan pada kolom `client_id` dan `timestamp`.

- **Autentikasi & Keamanan:** Skema OAuth2 dijalankan dengan bantuan pustaka misalnya **OAuthlib** atau fitur bawaan **FastAPI OAuth2PasswordBearer** (dimodifikasi untuk client_credentials). `client_id` dan `client_secret` disimpan di DB (terenkripsi/hashed). Token access yang diberikan berupa **JWT** (JSON Web Token) agar bisa diverifikasi tanpa memerlukan state di server (signature RSA256 diverifikasi pakai public key). JWT memuat claims seperti client_id dan exp (expiry). Dengan begitu, validasi token cepat dan tidak selalu perlu query DB (kecuali untuk verifikasi `client_secret` saat minting token). Tambahan HMAC signature untuk tiap request (seperti BCA API) bisa dipertimbangkan jika perlu keamanan ekstra, namun dengan HTTPS dan JWT sudah cukup di tahap awal.

- **Infrastructure:** Aplikasi dikontainerisasi menggunakan **Docker**. Satu container untuk komponen API+AI (bisa digabung atau terpisah tergantung skala). Nginx bisa digunakan sebagai reverse proxy di depan untuk meng-handle SSL termination dan static file (jika ada dokumentasi explorer). Sistem di-deploy di layanan cloud (misal AWS EC2 atau Heroku container). Untuk skalabilitas, dipersiapkan menggunakan **Kubernetes** agar bisa auto-scale pods berdasarkan load. Monitoring dilakukan dengan mengintegrasikan **Prometheus** (mengumpulkan metric seperti latensi, jumlah request, dll) dan dashboard **Grafana**.

- **Testing:** Pengembangan dilengkapi dengan **unit tests** dan **integration tests**. Contohnya, menggunakan `pytest` di Python untuk menguji fungsi analisis sentiment (diberi teks "I love it" harus keluar positive, dsb.) dan menguji endpoint dengan klien HTTP simulasi memastikan status code & output sesuai.

Berikut struktur direktori proyek (jika direpresentasikan dalam repository git):

```
herta-vibe-checker/
├── app/
│   ├── __init__.py
│   ├── main.py            # Entry point (API app init, route definitions)
│   ├── auth.py            # Module for OAuth2
│   ├── models.py          # DB models
│   ├── nlp_engine.py      # NLP analysis logic
│   ├── schemas.py         # Pydantic models for request/response schemas
│   ├── db.py              # Database connection setup
│   └── utils.py           # Utility functions (e.g., math challenge)
├── migrations/            # Database migration files (if using Alembic for s
chema)
├── tests/
│   ├── test_api.py        # Test cases for API endpoints
│   ├── test_nlp.py        # Test cases for NLP engine
│   └── ...
├── docker-compose.yml     # For local development (DB + app service)
├── Dockerfile             # Container image build for the API app
├── requirements.txt       # Python dependencies
└── README.md
```

Penjelasan ringkas: file `main.py` akan membuat objek FastAPI dan mendefinisikan route seperti `/health`, `/stats`, `/oauth/token`, `/vibe-check/single`, dll., dan menghubungkannya dengan fungsi handler yang memanggil modul lain. `auth.py` menangani pembuatan token JWT dan penyimpanan tantangan. `nlp_engine.py` memuat model AI (sekali saat startup) lalu menyediakan fungsi `analyze(text)` yang dipanggil tiap ada request vibe-check. `models.py` mendefinisikan struktur tabel sehingga bisa diakses via ORM (Object-Relational Mapping) – misal class `ClientApp` dengan property id, secret, dsb. serta class `VibeResult`. `db.py` menginisialisasi koneksi ke PostgreSQL (misal menggunakan SQLAlchemy URL yang diambil dari ENV).

Deployment

Setelah pengembangan, aplikasi dideploy di lingkungan server. Proses deployment dapat sebagai berikut:

1. **Container Build & Push:** CI pipeline (misal GitHub Actions) otomatis membangun Docker image setiap ada update kode. Image tersebut berisi semua dependency (Python env, model file yang diperlukan, dsb.). Setelah build sukses, image di-push ke registry (misal Docker Hub atau AWS ECR).

2. **Infrastructure Provisioning:** Menggunakan Kubernetes, dibuat sebuah Deployment untuk service API, mengatur jumlah replika awal (misal 3 pods). Juga service LoadBalancer untuk expose `api.herta-vibe.com`. Environment variable berisi config (DB URL, JWT secret/private key, etc.) diatur via ConfigMap/Secret.

3. **Database Setup:** PostgreSQL instance dijalankan (misal RDS di AWS) dan migrasi schema dijalankan (dari folder migrations) untuk membuat tabel. Credensial DB disimpan aman (Secret).

4. **Scaling & Monitoring:** Metrik dari pods dipantau. HPA (Horizontal Pod Autoscaler) diset untuk scale out jika CPU > X% (karena NLP mungkin intensif CPU) atau berdasarkan QPS. Log aplikasi dikumpulkan (misal ke Elasticsearch/Kibana) untuk dianalisa jika ada error runtime.

5. **Domain & SSL:** Domain `api.herta-vibe.com` dikonfigurasi mengarah ke load balancer. SSL certificate (LetsEncrypt atau AWS Certificate Manager) terpasang untuk TLS.

Setelah semua aktif, testing pasca-deploy dilakukan: memanggil `/health` memastikan OK, mencoba workflow auth + vibe-check di environment produksi.

Integrasi dengan Layanan API Lain

Untuk meningkatkan nilai bisnis, Herta's Vibe Checker dapat diintegrasikan dengan API eksternal milik pihak lain guna membangun solusi unik. Salah satu skenario integrasi yang diusulkan adalah menggabungkan *vibe checking* dengan data real-time dari platform media sosial, contohnya **integrasi dengan Twitter API**.

Contoh Integrasi (Herta's Vibe Checker + Twitter API):

Misalkan kita ingin layanan yang dapat memantau "vibe" publik terhadap suatu topik di Twitter secara real-time. Dengan integrasi ini, sistem akan mengambil tweet-tweet terbaru (misal melalui endpoint **Twitter API v2** yang menyediakan **filtered stream** atau **recent search**), lalu setiap tweet tersebut dilempar ke API Herta's Vibe Checker untuk dianalisis sentimennya. Hasil akhirnya bisa berupa dashboard atau notifikasi ketika sentimen publik berubah signifikan terhadap topik/keyword tertentu. Integrasi ini memadukan kemampuan **data stream** Twitter dengan **analisis sentimen** Herta's Vibe Checker, menghasilkan insight yang lebih kaya secara otomatis.

Langkah integrasinya:
1. **Setup Twitter API:** Daftarkan aplikasi di portal developer Twitter (sekarang dikenal sebagai X API). Dapatkan kredensial (API key & secret) dan lakukan autentikasi (Twitter mendukung OAuth2 Bearer Token untuk app-only access).
2. **Streaming/Fetching Tweets:** Gunakan endpoint seperti `/tweets/search/stream` untuk streaming realtime atau `/tweets/search/recent` untuk periodic fetch. Misal kita set filter untuk keyword tertentu (atau ambil semua tweet lalu filter sendiri).
3. **Call Vibe Checker API:** Setiap tweet (teks) yang masuk, kirim request ke endpoint `/vibe-check/single` Herta's API. Sertakan token Herta (didapat melalui proses OAuth2 Herta sebelumnya, bisa dijalankan sekali dan diperbarui tiap jam). Secara programmatic, ini bisa dilakukan dengan membuat worker asynchronous yang mengambil tweet dari stream, lalu memanggil API vibe check menggunakan HTTP client.
4. **Process Results:** Kumpulkan hasil vibe dari Herta's API. Misal out of 100 tweets terakhir tentang "GameX", 70% positif, 20% negatif, 10% netral. Informasi ini kemudian dapat ditampilkan di UI (misal grafik sentiment over time), atau digunakan untuk mengambil tindakan (misal jika mendeteksi banyak vibe negatif, trigger alert ke tim PR perusahaan).
5. **Unique Business Value:** Gabungan ini bisa menjadi **produk baru** – semacam *"Social Media Vibe Monitor"*. Misalnya dijual ke brand/marketer untuk memantau reaksi publik secara live saat mereka meluncurkan produk atau kampanye baru.

Integrasi di atas memerlukan penanganan beban cukup tinggi (karena stream tweet bisa sangat cepat). Untuk itu, bisa digunakan mekanisme **batching** sebelum memanggil vibe API (misal kumpulkan 10 tweet lalu gunakan endpoint batch `vibe-check/batch` untuk efisiensi). Juga penting mematuhi rate limit Twitter API dan Herta's API secara bersamaan.

Selain Twitter, integrasi dapat dilakukan dengan:
- **Twitch API atau Chat Bots:** Menghubungkan ke socket chat Twitch, kirim setiap N chat ke vibe checker untuk moderasi otomatis. Bot dapat secara otomatis me-*remove* atau

menandai pesan ber-“vibe” sangat negatif/toxic.
- **Discord Bot API:** Bot Discord menggunakan vibe checker untuk menganalisis mood obrolan di kanal komunitas.
- **YouTube Comments API:** Secara periodik scan komentar video, analisis vibenya, berikan laporan ke kreator tentang respons penonton.

Setiap integrasi membutuhkan kredensial dari platform terkait (API key/secret) dan mematuhi aturan penggunaan mereka. Misalnya, Twitter memiliki batasan kecepatan dan kebijakan data (data tweets tertentu tidak boleh disimpan permanen tanpa izin, dll.), sehingga sistem integrasi perlu mendesain *caching* dan penyimpanan dengan hati-hati (mungkin hanya simpan hasil agregat vibe, bukan isi tweet lengkap, untuk mematuhi kebijakan).

Skema OAuth2 Ganda: Dalam integrasi semacam ini, akan melibatkan dua lapis OAuth: OAuth untuk Twitter API (agar bisa mengambil tweet) dan OAuth untuk Herta’s API (untuk analisis vibe). Keduanya perlu dikelola. Aplikasi integrator akan menyimpan dua set token: Twitter Bearer Token dan Herta Bearer Token, masing-masing di header saat memanggil API terkait.

Kesimpulan

Dengan integrasi lintas API seperti contoh di atas, **Herta's Vibe Checker** memperluas jangkauan bisnisnya dari sekadar layanan analisis menjadi bagian dari solusi yang lebih besar. Dokumen ini menggambarkan sebuah contoh integrasi unik dengan **Twitter API milik Twitter (X)**. Hasil penggabungan ini dapat diwujudkan tanpa perlu mengubah fungsi dasar Herta's API – cukup membangun modul penghubung (middleware/adapter) yang mengambil data dari satu API, memproses via API lain.

Sebagai penutup, Herta's Vibe Checker dirancang modular dan aman sehingga **dapat dengan mudah digabungkan** dengan layanan pihak ketiga, menciptakan peluang inovasi di bidang analisis sentimen real-time untuk berbagai kebutuhan komunitas online. Dengan tiga dokumen yang telah disusun (dokumen bisnis, dokumentasi API, dan penjelasan implementasi+integrasi), diharapkan visi ide bisnis ini jelas dan siap untuk diimplementasikan serta dikembangkan lebih lanjut.