Implémentation de l'ID de Trace de Bout en Bout

Cet article de blog a été rédigé avec l'assistance de ChatGPT-4o.

J'ai travaillé sur une solution de trace ID de bout en bout pour garantir que chaque requête et réponse dans notre système puisse être suivie de manière cohérente entre le frontend et le backend. Cette solution facilite le débogage, la surveillance et la journalisation en associant chaque opération à un identifiant de trace unique. Voici une explication détaillée du fonctionnement de cette solution, accompagnée d'exemples de code.

Comment ça fonctionne

Frontend

La partie frontend de cette solution consiste à générer un identifiant de trace (trace ID) pour chaque requête et à l'envoyer, accompagné des informations du client, au backend. Cet identifiant de trace est utilisé pour suivre la requête à travers les différentes étapes de traitement sur le backend.

1. Collecte des informations client : Nous recueillons des informations pertinentes auprès du client, telles que les dimensions de l'écran, le type de réseau, le fuseau horaire, et plus encore. Ces informations sont envoyées avec les en-têtes de la requête.
2. Génération de l'ID de trace : Un ID de trace unique est généré pour chaque requête. Cet ID de trace est inclus dans les en-têtes de la requête, ce qui nous permet de suivre la requête tout au long de son cycle de vie.
3. API Fetch : La fonction `apiFetch` est utilisée pour effectuer des appels API. Elle inclut l'ID de trace et les informations du client dans les en-têtes de chaque requête.

Backend

La partie backend de la solution consiste à enregistrer l'ID de trace avec chaque message de journal et à inclure l'ID de trace dans les réponses. Cela nous permet de suivre les requêtes tout au long du traitement backend et de faire correspondre les réponses aux requêtes.

1. Gestion des Trace ID : Le backend reçoit le trace ID à partir des en-têtes de la requête ou en génère un nouveau s'il n'est pas fourni. Le trace ID est stocké dans un objet global Flask pour être utilisé tout au long du cycle de vie de la requête.
2. Journalisation : Des formateurs de logs personnalisés sont utilisés pour inclure l'ID de trace dans chaque message de log. Cela garantit que tous les messages de log liés à une requête peuvent être corrélés à l'aide de l'ID de trace.
3. Gestion des réponses : L'ID de trace est inclus dans les en-têtes de la réponse. Si une erreur se produit, l'ID de trace est également inclus dans le corps de la réponse d'erreur pour faciliter le débogage.

Kibana

Kibana est un outil puissant pour visualiser et rechercher des données de journaux stockées dans Elasticsearch. Avec notre solution de Trace ID, vous pouvez facilement suivre et déboguer des requêtes en utilisant Kibana. Le trace ID, qui est inclus dans chaque entrée de journal, peut être utilisé pour filtrer et rechercher des journaux spécifiques.

Pour rechercher des logs avec un ID de trace spécifique, vous pouvez utiliser le langage de requête Kibana (KQL). Par exemple, vous pouvez rechercher tous les logs liés à un ID de trace particulier avec la requête suivante :

```
trace_id:"Lc6t"
```

Cette requête retournera toutes les entrées de journal contenant l'ID de trace "Lc6t", vous permettant de suivre le chemin de la requête à travers le système. De plus, vous pouvez combiner cette requête avec d'autres critères pour affiner les résultats de recherche, comme filtrer par niveau de journal, horodatage, ou des mots-clés spécifiques dans les messages de journal.

En exploitant les capacités de visualisation de Kibana, vous pouvez également créer des tableaux de bord qui affichent des métriques et des tendances basées sur les identifiants de trace (trace IDs). Par exemple, vous pouvez visualiser le nombre de requêtes traitées, les temps de réponse moyens et les taux d'erreur, le tout corrélé avec leurs identifiants de trace respectifs. Cela aide à identifier des modèles et des problèmes potentiels dans les performances et la fiabilité de votre application.

L'utilisation de Kibana en conjonction avec notre solution de Trace ID offre une approche complète pour surveiller, déboguer et analyser le comportement de votre système, garantissant que chaque requête peut être efficacement suivie et investiguée.

Frontend

api.js

```
const BASE_URL = process.env.REACT_APP_BASE_URL;
```

// Fonction pour obtenir les informations du client const getClientInfo = () => { const { language, platform, cookieEnabled, doNotTrack, onLine } = navigator; const { width, height } = window.screen; const connection = navigator.connection || navigator.mozConnection || navigator.webkitConnection; const networkType = connection ? connection.effectiveType : 'unknown'; const timeZone = Intl.DateTimeFormat().resolvedOptions().timeZone; const referrer = document.referrer; const viewportWidth = window.innerWidth; const viewportHeight = window.innerHeight;

```
return {
    screenWidth: width,
    screenHeight: height,
    networkType,
    timeZone,
    language,
    platform,
    cookieEnabled,
    doNotTrack,
    onLine,
    referrer,
    viewportWidth,
    viewportHeight
};

};
```

// Fonction pour générer un identifiant de trace unique export const generateTraceId = (length = 4) => { const characters = 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789'; let traceId = ''; for (let i = 0; i < length; i++) { const randomIndex = Math.floor(Math.random() * characters.length); traceId += characters.charAt(randomIndex); } return traceId; };

```
export const apiFetch = async (endpoint, options = {}) => {
```

```
    const url = `${BASE_URL}${endpoint}`;
    const clientInfo = getClientInfo();

const traceId = options.traceId || generateTraceId();

const headers = {
    'Content-Type': 'application/json',
    'X-Client-Info': JSON.stringify(clientInfo),
    'X-Trace-Id': traceId,
    ...(options.headers || {})
};

const response = await fetch(url, {
    ...options,
    headers
});

    return response;
};
```

App.js

```
try {
  const response = await apiFetch('api', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
    },
    body: JSON.stringify(content),
    traceId: traceId
  });

if (response.ok) {
    const data = await response.json();
    //...
} else {
    const errorData = await response.json();
```

```
    const errorMessage = errorData.message || 'Une erreur inconnue s\'est produite';
    let errorToastMessage = errorMessage;
    errorToastMessage += ` (Trace ID: ${traceId})`;
    toast.error(errorToastMessage, {
      autoClose: 8000
    });
    setError(errorToastMessage);
}
} catch (error) {
  let errorString = error instanceof Error ? error.message : JSON.stringify(error);
```

const duration = (Date.now() - startTime) / 1000;

```
if (error.response) {
    // La requête a été faite et le serveur a répondu avec un code d'état en dehors de la plage 2xx
    errorString += ` (HTTP ${error.response.status}: ${error.response.statusText})`;
    console.error('Données d\'erreur de la réponse :', error.response.data);
} else if (error.request) {
    // La requête a été faite mais aucune réponse n'a été reçue
    errorString += ' (Aucune réponse reçue)';
    console.error('Données d\'erreur de la requête :', error.request);
} else {
    // Quelque chose s'est produit lors de la configuration de la requête qui a déclenché une erreur
    errorString += ` (Erreur lors de la configuration de la requête : ${error.message})`;
}
```

errorString += (ID de trace : ${traceId});

```
if (error instanceof Error) {
    errorString += `\nStack: ${error.stack}`;
}
```

errorString += JSON.stringify(error);

```
errorString += ` (Durée : ${duration} secondes)`;

toast.error(`Erreur : ${errorString}`, {
    autoClose: 8000
```

```
  });
  setError(errorString);
} finally {
  toast.dismiss(toastId);
}
```

Backend

__init__.py

```
# -*- encoding: utf-8 -*-

import os
import json
import time
import uuid
import string
import random

from flask import Flask, request, Response, g, has_request_context
from flask_cors import CORS

from .routes import initialize_routes
from .models import db, insert_default_config
import logging
from logging.handlers import RotatingFileHandler
from prometheus_client import Counter, generate_latest, Gauge
from flask_migrate import Migrate
from logstash_formatter import LogstashFormatterV1
```

app = Flask(**name**)

```
app.config.from_object('api.config.BaseConfig')

db.init_app(app)
initialize_routes(app)

CORS(app)
```

migrate = Migrate(app, db)

```
class RequestFormatter(logging.Formatter):
    def format(self, record):
        if has_request_context():
            record.trace_id = getattr(g, 'trace_id', 'unknown')
        else:
            record.trace_id = 'unknown'
        return super().format(record)

class CustomLogstashFormatter(LogstashFormatterV1):
    def format(self, record):
        if has_request_context():
            record.trace_id = getattr(g, 'trace_id', 'unknown')
        else:
            record.trace_id = 'unknown'
        return super().format(record)

def setup_loggers():
    logstash_handler = RotatingFileHandler(
        'app.log', maxBytes=100000000, backupCount=1)
    logstash_handler.setLevel(logging.DEBUG)
    logstash_formatter = CustomLogstashFormatter()
    logstash_handler.setFormatter(logstash_formatter)

txt_handler = RotatingFileHandler(
    'plain.log', maxBytes=100000000, backupCount=1)
txt_handler.setLevel(logging.DEBUG)
txt_formatter = RequestFormatter(
    '%(asctime)s %(levelname)s: %(message)s [in %(pathname)s:%(lineno)d] [trace_id: %(trace_id)s]')
txt_handler.setFormatter(txt_formatter)

    root_logger = logging.getLogger()
    root_logger.setLevel(logging.DEBUG)
    root_logger.addHandler(logstash_handler)
    root_logger.addHandler(txt_handler)
```

```
app.logger.addHandler(logstash_handler)
app.logger.addHandler(txt_handler)

werkzeug_logger = logging.getLogger('werkzeug')
werkzeug_logger.setLevel(logging.DEBUG)
werkzeug_logger.addHandler(logstash_handler)
werkzeug_logger.addHandler(txt_handler)

setup_loggers()

def generate_trace_id(length=4):
    characters = string.ascii_letters + string.digits
    return ''.join(random.choice(characters) for _ in range(length))

@app.before_request
def before_request():
    request.start_time = time.time()
    trace_id = request.headers.get('X-Trace-Id', generate_trace_id())
    g.trace_id = trace_id

    client_info = request.headers.get('X-Client-Info')
    if client_info:
        try:
            client_info_json = json.loads(client_info)
            logging.info(f"Informations du client : {client_info_json}")
        except json.JSONDecodeError:
            logging.warning("Format JSON invalide pour l'en-tête X-Client-Info")

@app.after_request
def after_request(response):
    response.headers['X-Trace-Id'] = g.trace_id

    if response.status_code != 200:
        logging.error(f'Code de statut de la réponse : {response.status_code}')
        logging.error(f'Corps de la réponse : {response.get_data(as_text=True)}')

        if response.content_type == 'application/json':
            try:
```

```
                response_json = response.get_json()
                response_json['trace_id'] = g.trace_id
                response.set_data(json.dumps(response_json))
            except Exception as e:
                logging.error(f"Erreur lors de l'ajout du trace_id à la réponse : {e}")

    return response
```

Journal

Vous pouvez rechercher tous les logs liés à un ID de trace spécifique avec la requête suivante :

```
trace_id:"Lc6t"
```


```
{
  "_index": "flask-logs-2024.07.05",
  "_type": "_doc",
  "_id": "Ae9zgZABqOMSOpxCZC5X",
  "_version": 1,
  "_score": 1,
  "_source": {
    "tags": [
      "_grokparsefailure"
    ],
    "filename": "generate.py",
    "funcName": "post",
    "message": "Requête traitée avec succès",
    "@version": 1,
    "name": "root",
    "host": "ip-172-31-35-xxx.ec2.internal",
    "relativeCreated": 685817.8744316101,
    "levelname": "INFO",
    "created": 1720158740.894831,
    "thread": 139715118360128,
    "threadName": "Thread-5",
    "levelno": 20,
```

```
    "pathname": "/home/project/project-name/api/routes/generate.py",
    "msecs": 894.8309421539307,
    "processName": "MainProcess",
    "lineno": 287,
    "path": "/home/project/project-name/app.log",
    "args": [],
    "source_host": "ip-172-31-35-xxx.ec2.internal",
    "module": "generate",
    "trace_id": "Lc6t",
    "stack_info": null,
    "process": 107613,
    "@timestamp": "2024-07-05T05:52:20.894Z"
  },
  "fields": {
    "levelname.keyword": [
      "INFO"
    ],
    "tags.keyword": [
      "_grokparsefailure"
    ],
    "relativeCreated": [
      685817.9
    ],
    "processName.keyword": [
      "MainProcess"
    ],
    "filename.keyword": [
      "generate.py"
    ],
    "funcName": [
      "post"
    ],
    "path": [
      "/home/project/project-name/app.log"
    ],
    "processName": [
```

```
    "MainProcess"
  ],
  "@version": [
    1
  ],
  "host": [
    "ip-172-31-35-xxx.ec2.internal"
  ],
  "msecs": [
    894.83093
  ],
  "source_host.keyword": [
    "ip-172-31-35-xxx.ec2.internal"
  ],
  "host.keyword": [
    "ip-172-31-35-xxx.ec2.internal"
  ],
  "levelname": [
    "INFO"
  ],
  "process": [
    107613
  ],
  "threadName.keyword": [
    "Thread-5"
  ],
  "trace_id": [
    "Lc6t"
  ],
  "source_host": [
    "ip-172-31-35-xxx.ec2.internal"
  ],
  "created": [
    1720158700
  ],
  "module": [
```

```
    "generate"
  ],
  "module.keyword": [
    "generate"
  ],
  "name.keyword": [
    "root"
  ],
  "thread": [
    139715118360128
  ],
  "message": [
    "Requête traitée avec succès"
  ],
  "levelno": [
    20
  ],
  "trace_id.keyword": [
    "Lc6t"
  ],
  "threadName": [
    "Thread-5"
  ],
  "pathname": [
    "/home/project/project-name/api/routes/generate.py"
  ],
  "tags": [
    "_grokparsefailure"
  ],
  "pathname.keyword": [
    "/home/project/project-name/api/routes/generate.py"
  ],
  "@timestamp": [
    "2024-07-05T05:52:20.894Z"
  ],
  "filename": [
```

```
      "generate.py"
    ],
    "lineno": [
      287
    ],
    "message.keyword": [
      "Requête traitée avec succès"
    ],
    "name": [
      "root"
    ],
    "funcName.keyword": [
      "post"
    ],
    "path.keyword": [
      "/home/project/project-name/app.log"
    ]
  }
}
```


Comme indiqué ci-dessus, vous pouvez voir l'ID de trace dans le journal.