Kean Nafis S.

Critiquill Service

Date: 14/11/2023 **Time:** 07:00 AM **NIM:** 18221148

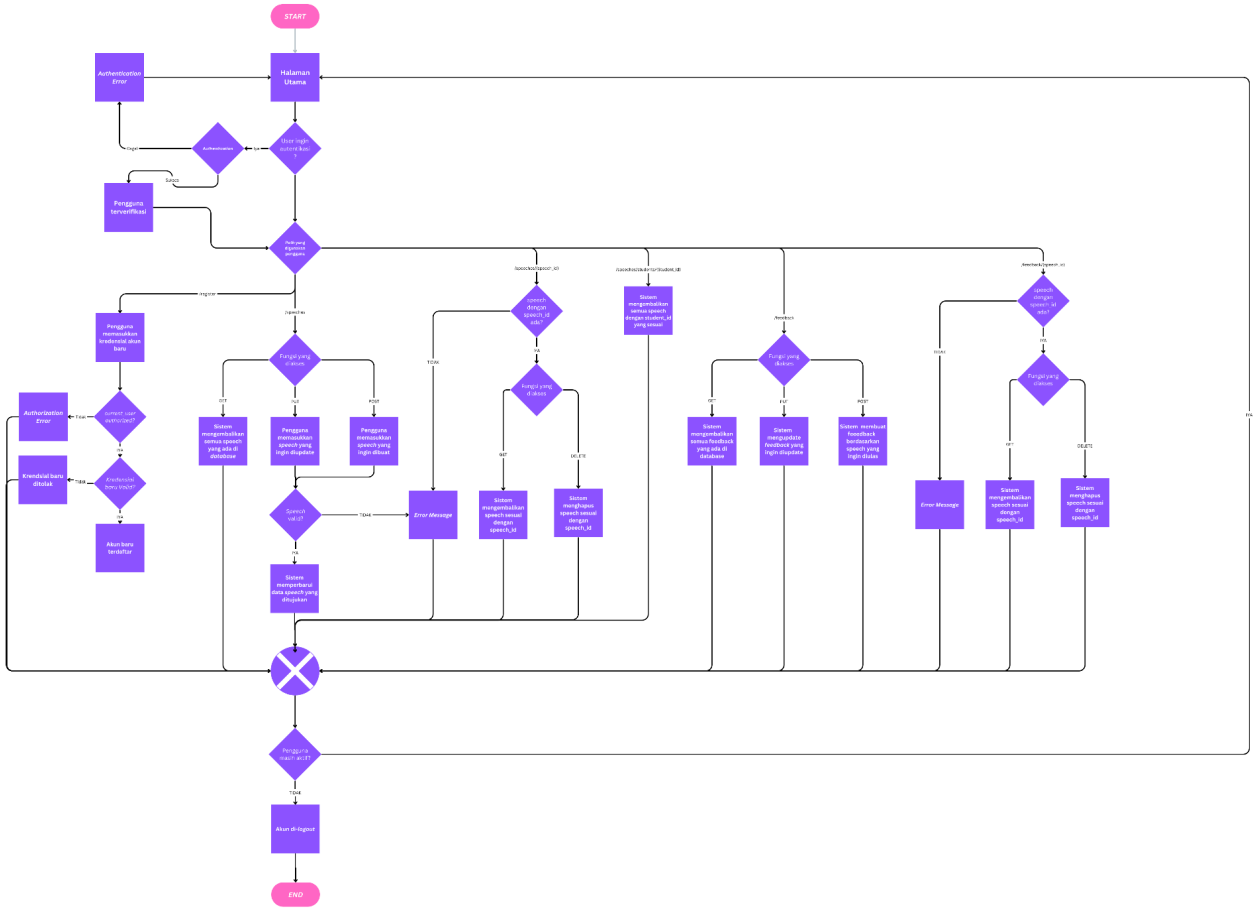
Apa itu CritiQuill?

Critiquill adalah *microservice* yang **meringkas pidato debat** ke dalam format yang telah ditentukan dan secara otomatis **menghasilkan umpan balik untuk pidato tersebut**. Formulir umpan balik yang dihasilkan dibuat menggunakan algoritma yang mengevaluasi kualitas pidato secara keseluruhan dan dengan memprioritaskan aspek yang paling penting. Critiquill juga menulis pidato dan umpan balik yang dihasilkannya ke dalam file JSON. Umpan balik yang dihasilkan oleh layanan akan menilai setiap aspek pidato yang disampaikan dan memberikan umpan balik yang mengutamakan aspek terpenting yang masih kurang baik.

Fungsionalitas utama dari Critiquill adalah pembuatan *feedback* atau umpan balik untuk sebuah pidato debat secara otomatis. Pidato atau *speech* yang digunakan sebagai bahan ulasan harus disimpan di *database* critiquill terlebih dahulu. Critiquill membuat *feedback* dengan menganalisis atribut yang disimpan di data *speech* dan menggunakan sebuah algoritma untuk menentukan kualtias *speech*-nya dan hal apa saja yang dapat diperbaiki.\

Cara Kerja

Link Flowchart: https://bit.ly/CritiquillFlowchart



Fungsi dan Endpoint yang Dapat Diakses

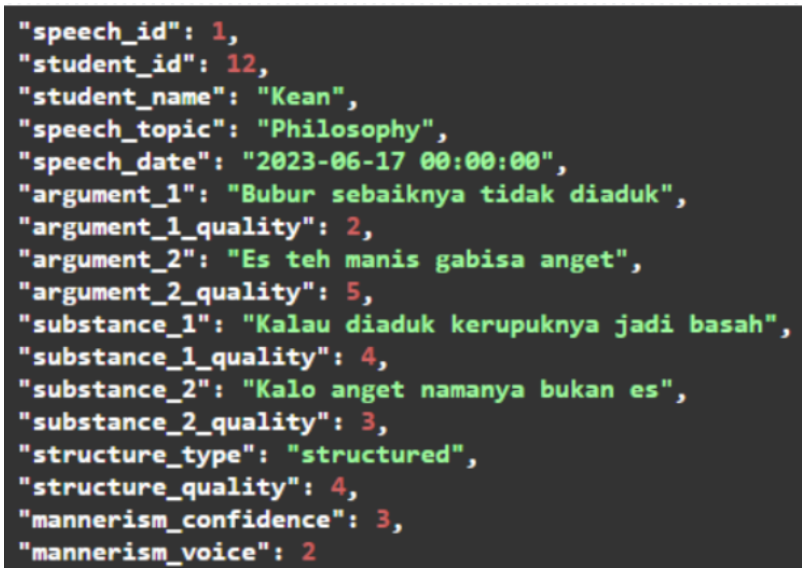
Path	CRUD	Fungsi
/register	**POST**	Menambahkan akun *user* baru ke *database*
/speeches	**GET**	Mengambil semua *speech* yang terdata di dalam *database*
/speeches	**PUT**	Memperbarui data *speech* yang sudah ada
/speeches	**POST**	Menambahkan *speech* baru ke dalam *database*
/speeches/students/{student_id}	**GET**	Mengembalikan semua *speech_id* yang terdata di dalam *database* yang dibuat oleh murid dengan *student_id* yang sesuai
/speeches/{speech_id}	**GET**	Mengembalikan *speech* yang terdata di dalam *database* yang memiliki speech_id yang sesuai
/speeches/{speech_id}	**DELETE**	Menghapus *speech* yang terdata di dalam *database* yang memiliki speech_id yang sesuai
/feedback	**GET**	Mengambil semua data *feedback* dari *database*
/feedback	**PUT**	Memperbarui data sebuah *feedback* yang ada di *database*
/feedback	**POST**	Generate sebuah feedback secara otomatis berdasarkan *speech* yang sudah disimpan
/feedback/{feedback_id}	**GET**	Mengambil data sebuah *feedback* dari database sesuai dengan feedback_id
/feedback/{feedback_id}	**DELETE**	Menghapus data sebuah *feedback* dari database sesuai dengan feedback_id

Pembuatan Speech dan Feedback

1. Sebelum Anda bisa mendapatkan masukan atas pidato debat Anda, Anda perlu mengirimkan pidato debat Anda ke Critiquill untuk disimpan. Untuk membuat prosesnya efisien, Critiquill telah menyediakan format standar kepada tutor untuk semua transkrip pidato Anda. Setiap transkrip pidato harus diubah ke dalam format berikut:

Atribut	Keterangan
speech_id	Integer yang mewakilkan sebuah ID pidato yang unik
student_id	Integer yang mewakilkan ID murid yang membuat pidato yang dimasukkan
student_name	String yang mewakilkan nama murid yang membuat pidato yang dimasukkan
speech_topic	String yang mewakilkan topik atau tema utama debat dari pidato yang dimasukkan
speech_date	String (diubah dari format datetime) yang mewakilkan tanggal dan waktu pembuatan pidato yang dimasukkan
argument_1	String yang mengandung intisari dari argumen pertama di pidato yang dibuat
argument_1_quality	Integer (dari 1-5) yang mewakilkan tingkat kualitas argumen pertama di pidato yang dimasukkan
argument_2	String yang mengandung intisari dari argumen kedua di pidato yang dibuat
argument_2_quality	Integer (dari 1-5) yang mewakilkan tingkat kualitas argumen kedua di pidato yang dimasukkan
substance_1	String yang berisi detail dari argumen pertama pada pidato yang dimasukkan
substance_1_quality	Integer (dari 1-5) yang mewakilkan tingkat kualitas substance pertama di pidato yang dimasukkan
substance_2	String yang berisi detail dari argumen kedua pada pidato yang dimasukkan
substance_2_quality	Integer (dari 1-5) yang mewakilkan tingkat kualitas substance kedua di pidato yang dimasukkan
structure_type	String yang berisi jenis struktur pidato yang digunakkan di dalam debat
structure_quality	Integer (dari 1-5) yang mewakilkan tingkat kualitas struktur pidato yang digunakan
mannerism_confidence	Integer (dari 1-5) yang mewakilkan tingkat kualitas kepercayaan diri dari pidato yang digunakan
mannerism_voice	Integer (dari 1-5) yang mewakilkan tingkat kualitas penggunaan volume, tempo, dan intonasi suara dari pidato yang digunakan

Contoh:



```
"speech_id": 1,
"student_id": 12,
"student_name": "Kean",
"speech_topic": "Philosophy",
"speech_date": "2023-06-17 00:00:00",
"argument_1": "Bubur sebaiknya tidak diaduk",
"argument_1_quality": 2,
"argument_2": "Es teh manis gabisa anget",
"argument_2_quality": 5,
"substance_1": "Kalau diaduk kerupuknya jadi basah",
"substance_1_quality": 4,
"substance_2": "Kalo anget namanya bukan es",
"substance_2_quality": 3,
"structure_type": "structured",
"structure_quality": 4,
"mannerism_confidence": 3,
"mannerism_voice": 2
```

2. Setelah Anda mengisi format pidato yang disediakan, gunakan metode POST pada jalur /speeches untuk menambahkan pidato ke dalam file JSON. Setiap ucapan harus memiliki ID_ucapan yang unik agar *response*-nya valid

```
curl -X 'POST' \
  'http://20.92.28.205/speeches' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
  "speech_id": 4,
  "student_id": 21,
  "student_name": "Santang",
  "speech_topic": "Indonesian Politics",
  "speech_date": "2023-02-02 00:00:00",
  "argument_1": "Lorem ipsum",
  "argument_1_quality": 3,
  "argument_2": "Lorem ipsum",
  "argument_2_quality": 3,
  "substance_1": "Lorem ipsum",
  "substance_1_quality": 3,
  "substance_2": "Lorem ipsum",
  "substance_2_quality": 3,
  "structure_type": "Lorem ipsum",
  "structure_quality": 3,
  "mannerism_confidence": 3,
  "mannerism_voice": 3
}'
```

3. Setelah Anda mengirimkan pidato ke dalam sistem (atau jika pidato yang ingin Anda beri umpan balik sudah ada), gunakan metode POST di jalur /feedback untuk menghasilkan umpan balik untuk pidato tertentu. Setiap pidato hanya dapat memiliki satu umpan balik, jadi program tidak akan berfungsi jika pidato yang ingin Anda hasilkan umpan baliknya

sudah memiliki formulir umpan balik. Bila menggunakan metode tersebut, Anda akan diminta untuk mengisi speech_id dari pidato yang anda ingin buat umpan balik-nya

4. Setelah Anda berhasil melakukan langkah 3, umpan balik baru akan dibuat secara otomatis di file JSON, setiap penerbitan formulir umpan balik akan mengikuti format yang sama
5. Untuk membaca umpan balik tertentu, gunakan metode GET pada jalur /feedback/{speech_id} untuk membaca masukan yang ada berdasarkan speech_id yang Anda masukkan sebagai parameter

Response body


```
{
  "speech_id": 3,
  "student_id": 15,
  "score": 25,
  "area_of_improvement_1": "Work on analyzing and improving your first argument",
  "overall_feedback": "Great Speech! You have solid fundamentals and good execution. Work on taking it to the next level",
  "area_of_improvement_2": "Work on adding weight adding substance to your second argument"
}
```

Critiquill menggunakan JWT (JSON Web Token) dan OAuth 2.0 untuk proses *authentication* dan *authorization* di dalam layanan Critiquill. OAuth 2.0 digunakan sebagai dasar untuk *authorization* pengguna ketika ingin mengakses layanan Critiquill dan sebagai dasar mekanisme *authentication* dan *authorization*. JWT (JSON Web Token) digunakan untuk membuat token dan untuk *decode* token

OAuth 2.0 memungkinkan aplikasi pihak ketiga untuk mendapatkan akses terbatas kepada sebuah layanan berbasis HTTP. OAuth 2.0 dapat memberikan akses dengan mewakilkan sebuah *resource owner* atau dengan memberikan izin kepada aplikasi pihak ketiga untuk mengakses layanan yang diinginkan. OAuth 2.0 menggunakan sistem yang melibatkan tiga pihak: *resource owner (*user), *client (application)*, dan *resource server* (API).

JWT atau JSON Web Token terdiri dari tiga komponen, yaitu *header*, *payload*, dan *signature*. *Header* mengandung metadata mengenai tokennya. *Payload* mengandung data yang ingin di-*encode* ke dalam tokennya. Semakin banyak data yang ingin disimpan, semakin besar JWT yang dihasilkan. *Signature* menggunakan *header, payload*, dan sebuah secret yang disimpan di dalam server. *Signature* digunakan untuk memastikan akses terhadap token dilakukan dengan aman dan terjaga.

OAuth 2.0 dengan sendirinya tidak cukup untuk melakukan *authentication*. Untuk melakukan *authentication*, OAuth 2.0 dibantu JWT (JSON Web Token). JWT bertanggung jawab atas autentikasi pengguna yang ingin mengakses layanan Critiquill dan OAuth 2.0 digunakan untuk *authorization. Access token* yang disediakan oleh OAuth 2.0 menggunakan bentuk JWT. JWT mengandung informasi mengenai *scope* (akses yang diberikan), *client ID*, *user ID*, dan informasi relevan lainnya.

Pada OAuth 2.0 yang diimplementaasikan di Critiquill. *Resource server* butuh mem-validasi JWT yang diterima sebagai *access token*. Pada proses ini, *resource server* memeriksa *signatrue, expiration,* dan informasi lain dari token yang diterima. Seluruh proses validasi dilakukan oleh *resource server* karena JWT bersifat *self-contained*.

Kelas Token dan TokenData

Untuk melakukan proses pembuatan dan pemrosesan Token, Critiquill membuat dua kelas, yaitu kelas Token yang mengandung *access_token* dan *token_type* dan TokenData yang mengandung username dari Token yang di-*generate* untuk user. Kedua kelas menggunakan BaseModel untuk memudahkan proses *building*

```
class Token(BaseModel):
    access_token: str
    token_type: str

class TokenData(BaseModel):
    username: str or None = None
```

Variabel untuk JWT

- SECRET_KEY adalah variabel yang akan dihasilkan untuk membantu pemrosesan token supaya *secure*
- ALGORITHM adalah variabel yang digunakan untuk menentukan algoritma validasi *signature* . variabel ini memiliki nilai HS256 yaitu sebuah algoritma simetris yang membagi satu *secret key* di antara *identity provider* dan aplikasi Critiquill
- ACCESS_TOKEN_EXPIRES_MINUTES adalah variabel yang menentukan durasi sebelum *access token* tidak dapat digunakan lagi. Pada Critiquill, nilai ini adalah 10 (menit).

```
SECRET_KEY = ""
ALGORITHM = "HS256"
ACCESS_TOKEN_EXPIRE_MINUTES = 15
```

Instansiasi Kelas CryptContext

CryptContext adalah sebuah kelas yang didapatkan dari library Passlib untuk mengelola *password hashing* dan verifikasi. Kelas CryptContext menerima sebuah skema hashing yang ingin digunakan. Pada Critiquill, skema hashing yang digunakan adalah bcrypt. Kelas CryptContext juga menerima pengaturan deprekasi *hashing scheme* yang sudah *outdated* atau tidak *secure.* Semua hal tersebut dilakukan pada cuplikan kode tersebut. Instansiasi CryptContext tersebut disimpan dalam variabel pwd_context.

```
pwd_context = CryptContext(schemes=["bcrypt"], deprecated="auto")
```

Instansiasi OAuth2PasswordBearer

OAuth2PasswordBearer adalah kelas yang disediakan di dalam library FastAPI untuk mengurus autentikasi berbasis OAuth 2.0. skema hashing yang digunakan juga bcrypt, sama seperti CryptContext. tokenUrl adalah atribut yang menentukan URL yang akan digunakan *client* untuk mengirim *token request*. Untuk Critiquill, URL tersebut adalah "/token". *Client* mengirim sebuah perintah POST ke endpoint "/token" dengan kredensial yang sesuai. Instansiasi OAuth2PasswordBearer disimpan ke dalam variabel oauth_2_scheme.

```
oauth_2_scheme = OAuth2PasswordBearer(tokenUrl= 'token')
```

Fungsi untuk membuat password hash

Untuk mengubah password yang masih berupa plaintext, misal "password123", menjadi digest dari hash, digunakan method CryptContext yaitu hash() yang menerima parameter password yang ingin di-hash. Hasil hashing tersebut dikembalikan di akhir fungsi.

```
def get_password_hash(password):
    return pwd_context.hash(password)
```

Fungsi untuk verifikasi password

Untuk membandingkan password yang dimasukkan dengan password yang disimpan (yang sudah di-hash), digunakan method bawaan CryptContext yaitu verify yang menerima parameter password dalam bentuk plaintext dan password yang dijadikan hash.

```
def verify_password(plain_password, hashed_password):
    return pwd_context.verify(plain_password, hashed_password)
```

Autentikasi User

Fungsi authentiate_user digunakan untuk memverifikasi kredensial user yang dimasukkan. Fungsi ini menerima parameter dataset pengguna, username yang dimasukkan untuk diverifikasi, dan password yang ingin diverifikasi. Pertama, fungsi ini mencari username di data user yang disediakan. Bila ditemukan user yang sesuai, fungsi authenticate_user akan memanggil verify_password untuk membandingkan password yang dimasukkan dengan hashed password yang disimpan di database pengguna.

```
def authenticate_user(userlist: dict, username: str, password: str):
    user = get_user(userlist, username)
    if not user:
        return False
    if not verify_password(password, user.password_hash):
        return False

    return user
```

Membuat (Encode) Access Token

Fungsi create_access_token digunakan untuk meng-encode sebuah *access token* yang akan digunakan pada proses-proses autentikasi dan otorisasi lainnya. Fungsi ini memiliki sebuah parameter yang menerima sekumpulan data dari user dan waktu ekspirasi token.

```
def create_access_token(data: dict, expires_delta: timedelta or None = None):
    to_encode = data.copy()
    if expires_delta:
        expire = datetime.utcnow() + expires_delta
    else:
        expire = datetime.utcnow() + timedelta(minutes = 10)

    to_encode.update({"exp": expire})
    encoded_jwt = jwt.encode(to_encode, SECRET_KEY, algorithm=ALGORITHM)
    return encoded_jwt # the access token
```

Verifikasi Current User

Fungsi get_current_user digunakan untuk meng-*decode* sebuah token, verifikasi user yang sedang aktif, dan memproses token dari user tersebut. Payload ditentukan dengan jwt.decode yang menerima token dari oauth_2_scheme, SECRET_KEY, dan algoritma HS256 yang telah didefinisikan sebelumnya. Payload tersebut digunakan untuk menentukan username yang memiliki token tersebut. Bila ditemukan sebuah user, tokend_data diekstrak dari data user tersebut. Hasil akhir dari fungsi ini adalah user yang telah diverifikasi menggunakan token JWT

```python
# AUTHORIZATION PROCESS
async def get_current_user(token: str = Depends(oauth_2_scheme)):
    credential_exception = HTTPException(status_code = status.HTTP_401_UNAUTHORIZED,
                                         detail="coud not validate user credentials", headers={"WWW-Authenticate": "Bearer"})
    try:
        # parse out the token and decode it
        payload = jwt.decode(token, SECRET_KEY, algorithms=[ALGORITHM])
        username: str = payload.get("sub")
        # check if the user exists
        if username is None:
            raise credential_exception
        # get the user data
        token_data = TokenData(username = username)
    except JWTError:
        raise credential_exception

    # checks if the user exists in the database
    user = get_user(users_data, username = token_data.username)

    if user is None:
        raise credential_exception

    return user
```

Pembuatan Token

Path "/token" dibuat sebagai endpoint untuk pembuatan access token. Endpoint ini mengandung fungsi login_for_access_token dengan parameter yang menerima sebuah OAuth2PasswordRequestForm sebagai prasyarat keberjalanan autentikasi. Bila form autentiaksi sudah diisi, fungsi ini akan memanggil authenticate_user untuk memverifikasi kredensial yang diterima dan menyediakan access token untuk pengguna tersebut.

```python
# writing tokens
@app.post("/token", response_model=Token)
async def login_for_access_token(form_data: OAuth2PasswordRequestForm = Depends()):
    user = authenticate_user(users_data, form_data.username, form_data.password)
    if not user:
        raise HTTPException(status_code=status.HTTP_401_UNAUTHORIZED, detail="Incorrect username or password",
                            headers={"WWW-Authenticate": "Bearer"})

    access_token_expires = timedelta(minutes= ACCESS_TOKEN_EXPIRE_MINUTES)
    access_token = create_access_token(data={"sub": user.username}, expires_delta= access_token_expires)
    return {"access_token": access_token, "token_type": "bearer"}
```