System Design – Video Editor Architecture

Owen Christian Cahyadi | owenchristian123@gmail.com

1. High-Level Overview :
    1.1. Provide a diagram or a text-based explanation of the key components (front-end and back-end, if any) and their interactions.
        1.1.1. **Frontend (Client-Side)**:
            1.1.1.1. **Browser**: Antarmuka utama yang digunakan oleh pengguna untuk mengakses aplikasi video editor berbasis web. Melalui browser, pengguna dapat mengunggah video, mengeditnya, dan melihat hasil pengeditan. Pengguna tidak perlu menginstal perangkat lunak tambahan, sehingga memberikan kemudahan akses dari berbagai perangkat tanpa pembatasan platform. Selain itu, browser modern mendukung berbagai API seperti WebRTC untuk komunikasi waktu nyata dan WebAssembly untuk pemrosesan video.
            1.1.1.2. **Video Player**: Komponen yang memungkinkan pengguna untuk memutar dan melihat video yang sedang diedit dalam editor. Ini termasuk fungsi seperti play, pause, seek (menyisir waktu video), dan pengaturan volume. Video player merupakan komponen dasar yang memungkinkan pengguna untuk melihat hasil pengeditan secara langsung, termasuk penambahan efek atau transisi. Ini sangat penting untuk pengalaman pengguna, karena pengeditan video membutuhkan feedback visual secara langsung.
            1.1.1.3. **Canvas & Timeline Editor**: Area tempat video dan elemen lainnya ditampilkan secara visual. Timeline editor memungkinkan pengguna untuk memanipulasi video dengan mengatur durasi, urutan, dan efek yang diterapkan. Pengguna dapat drag-and-drop klip video, menambahkan efek visual, dan transisi di sepanjang timeline. Canvas dan timeline adalah jantung dari pengeditan video. Mereka memberikan antarmuka yang diperlukan untuk memanipulasi elemen-elemen video secara presisi. Tanpa fitur ini,

pengguna akan kesulitan mengedit video dengan cara yang efisien dan intuitif.

1.1.1.4. **API Requests**: Frontend mengirimkan permintaan API ke backend untuk mendapatkan metadata video (misalnya, durasi, format) atau status pengeditan. Selain itu, API digunakan untuk menyimpan status pengeditan dan mengambil hasil akhir video yang sudah diproses. Komunikasi antara frontend dan backend sangat penting untuk memastikan data yang dibutuhkan selalu tersedia. API memungkinkan frontend dan backend berinteraksi secara terstruktur dan memungkinkan penyimpanan metadata, serta memproses data video secara asinkron.

1.1.1.5. **WebSocket/RTC**: WebSocket digunakan untuk komunikasi dua arah secara real-time antara frontend dan backend, misalnya untuk live collaboration atau mengupdate status pengeditan secara langsung. WebRTC juga bisa digunakan untuk streaming video langsung antar pengguna. Dalam konteks kolaborasi atau fitur real-time (misalnya, banyak pengguna mengedit video secara bersamaan), WebSocket atau WebRTC memungkinkan data atau perubahan yang terjadi pada video langsung terlihat oleh pengguna lain tanpa perlu memuat ulang halaman.

1.1.2. **Backend (Server-Side)**:

1.1.2.1. **API Server**: Berfungsi untuk menerima dan memproses permintaan dari frontend, seperti permintaan untuk mengambil metadata video, status pengeditan, atau hasil pemrosesan video. Ini juga mengatur autentikasi dan otorisasi pengguna. API server mengelola logika aplikasi dan alur data antar komponen. Ia bertindak sebagai penghubung antara frontend dan berbagai layanan lainnya di backend (misalnya, video processing engine, database, dll.). Tanpa API server, tidak akan ada komunikasi yang terstruktur antara klien dan server.

1.1.2.2. **Video Processing Engine**: Engine pemrosesan video digunakan untuk menangani tugas-tugas berat seperti transcoding, rendering, dan encoding video. Ini akan memproses video yang diunggah

oleh pengguna, menghasilkan video yang telah diedit dan siap didistribusikan. Pengeditan video adalah tugas yang sangat berat secara komputasi, terutama jika menyangkut pemrosesan file video dengan resolusi tinggi. Oleh karena itu, komponen ini sangat penting untuk menangani pemrosesan ini di server terpisah untuk mencegah overload pada API server dan frontend.

1.1.2.3. **Authentication Server**: Server autentikasi mengelola identitas pengguna, menyediakan login, logout, dan sistem otorisasi berbasis token. Keamanan adalah aspek yang sangat penting dalam aplikasi apa pun yang menyimpan data pengguna, terutama yang melibatkan file video yang sensitif atau berharga. Authentication server memastikan bahwa hanya pengguna yang sah yang dapat mengakses data dan fitur tertentu dari aplikasi.

1.1.2.4. **Job Queue**: Antrian pekerjaan digunakan untuk memproses tugas yang membutuhkan waktu lama, seperti rendering atau transcoding video, secara asinkron. Permintaan dari frontend akan dimasukkan ke dalam antrian dan diproses secara berurutan oleh worker. Dengan adanya job queue, server backend dapat menangani tugas pemrosesan berat tanpa mengganggu pengalaman pengguna secara langsung. Ini memungkinkan aplikasi tetap responsif, bahkan jika ada tugas pemrosesan yang intensif.

1.1.3. **Database/Metadata**:

1.1.3.1. **Relational Database (SQL)**: Menyimpan metadata video (seperti nama file, durasi, format, status edit, dll.) dan informasi pengguna (akun, histori pengeditan).

1.1.3.2. Storage(Cloud): Digunakan untuk menyimpan file video yang diunggah dan hasil video yang sudah diproses. Ini termasuk penyimpanan untuk video input, video output, dan file-file terkait seperti thumbnail dan metadata. Mengingat ukuran video yang besar, penyimpanan yang efisien dan terkelola sangat penting. Penyimpanan cloud memberikan fleksibilitas, skalabilitas, dan

keandalan dalam mengelola file video, sehingga aplikasi dapat menangani video dengan ukuran besar tanpa masalah kapasitas.

1.1.4. **Content Delivery Network (CDN)**:

CDN digunakan untuk mendistribusikan video yang diunggah dan video yang sudah diproses ke berbagai lokasi server global, sehingga pengguna dapat mengakses file dengan latensi yang lebih rendah, bahkan jika mereka berada di lokasi yang jauh dari server utama. Tanpa CDN, distribusi video akan menjadi sangat lambat, terutama jika ada banyak pengguna yang mengakses konten dari lokasi geografis yang tersebar. CDN memastikan bahwa video dapat diakses dengan cepat dan efisien dari berbagai tempat di dunia, meningkatkan pengalaman pengguna secara keseluruhan.

1.1.5. **Cache (Server-Side & Client-Side)**:

1.1.5.1. **Server-side cache**: Menyimpan data yang sering diakses oleh pengguna, seperti metadata video atau thumbnail, untuk mengurangi waktu respons server.

1.1.5.2. **Client-side cache**: Menyimpan data di sisi klien (browser) untuk mempercepat akses berikutnya ke data yang sama tanpa perlu mengirim permintaan ke server.

1.1.6. **Load Balancer**:

Load balancer mendistribusikan trafik dan permintaan API secara merata ke berbagai server backend. Ini bertujuan untuk mencegah satu server menjadi kelebihan beban dan memastikan skalabilitas aplikasi. Load balancer memastikan ketersediaan tinggi dan mengoptimalkan penggunaan sumber daya server. Tanpa load balancer, jika satu server terlalu banyak menangani permintaan, aplikasi dapat melambat atau bahkan gagal melayani pengguna dengan benar. Ini juga membantu dalam menangani lonjakan trafik atau kegagalan server dengan mengalihkan permintaan ke server yang lebih sehat.

1.1.7. **Internet Protocols (TCP)**:

Protokol ini digunakan untuk komunikasi antara klien dan server. TCP (Transmission Control Protocol) digunakan untuk komunikasi yang andal dan terjamin (misalnya, saat mengirim dan menerima data atau video). TCP memastikan bahwa data yang dikirim antar server dan klien tiba dengan aman dan dalam urutan yang benar, yang sangat penting untuk file besar seperti video.

1.1.8. **Interaksi Antar Komponen:**

1.1.8.1. **User(Browser)**: Pengguna mengunggah file video melalui File Input di antarmuka pengguna (browser). Video yang diunggah akan dikirim ke server backend menggunakan permintaan HTTP POST. Pengguna juga dapat melakukan permintaan API untuk mengambil data video yang telah diproses, metadata, atau hasil pengeditan.

1.1.8.2. **Video Player**: Mengambil file video dari File Storage untuk diputar di browser. Video yang diputar dapat dipause, diserongkan, atau dipercepat. Video yang sedang diputar dikirimkan melalui Content Delivery Network (CDN) untuk mengurangi latensi, terutama saat melayani pengguna dari lokasi geografis yang jauh.

1.1.8.3. **Timeline Editor**: Frontend akan memperbarui tampilan Timeline sesuai dengan input pengguna (misalnya, drag-and-drop klip). Backend mendapatkan data tentang urutan dan durasi klip dari API Requests untuk memulai pemrosesan pengeditan. Data timeline yang telah disesuaikan akan dikirim ke server untuk diproses lebih lanjut dan disimpan di Database. Setelah pengeditan selesai, hasilnya dikirim kembali ke Browser untuk ditampilkan.

1.1.8.4. **Load Balancer**: Load balancer menerima permintaan dari pengguna dan mendistribusikannya ke server backend yang paling tersedia atau dengan beban paling rendah. Dalam konteks penyimpanan video, load balancer juga dapat mendistribusikan permintaan pengambilan video ke server yang lebih dekat dengan lokasi pengguna atau yang memiliki data video yang diperlukan.

1.1.8.5. **Backend (Server-side)**: API Requests dari frontend (browser) diterima dan diproses oleh backend. Misalnya, ketika video diunggah, backend akan menerima file dan menyimpannya di File Storage atau Database. Backend juga bertanggung jawab untuk memproses video (misalnya, menerapkan filter atau efek transisi) dengan menggunakan berbagai alat atau pustaka pemrosesan video. Metadata yang relevan mengenai video (seperti durasi, format, atau data pengeditan) disimpan dalam Database dan diambil setiap kali diperlukan untuk menampilkan data atau hasil pengeditan. Backend mengelola Authentication Server untuk memastikan bahwa hanya pengguna yang sah yang dapat mengakses atau mengedit video mereka.

1.1.8.6. **API Request (HTTP Request)**: GET Requests, Frontend mengirimkan permintaan GET untuk mengambil metadata video, status pengeditan, atau hasil video yang sudah selesai. POST Requests, untuk mengunggah video atau memperbarui status pengeditan, frontend mengirimkan permintaan POST untuk menyimpan atau mengubah data di server. API dapat digunakan untuk melakukan pembaruan metadata dan status pengeditan yang kemudian akan disimpan di Database.

1.1.8.7. **File Storage (Cloud)**: File Uploading, video yang diunggah oleh pengguna disimpan dalam File Storage setelah diterima oleh backend. Video Processing, backend memproses file video yang disimpan dan menghasilkan versi yang telah diedit yang kemudian disimpan kembali di File Storage. Video Retrieval, video yang telah disimpan dapat diambil kembali oleh frontend untuk diputar dalam Video Player.

1.1.8.8. **Database/Metadata**: Metadata video, seperti durasi dan format, disimpan dalam Database untuk diambil saat diperlukan, misalnya untuk menampilkan informasi file atau untuk memanipulasi data pengeditan. Pada status pengeditan, setiap perubahan atau penyesuaian yang dilakukan pada video (seperti penambahan transisi atau efek) disimpan dalam Database untuk diambil kembali saat video diproses. Pada Query Requests, Frontend

mengirimkan permintaan untuk mengambil atau memperbarui data terkait video atau pengeditan melalui API Requests yang kemudian diproses oleh backend.

1.1.8.9. **CDN**: Video yang telah disimpan dalam File Storage dapat dipindahkan atau disalin ke CDN untuk akses yang lebih cepat oleh pengguna di lokasi yang lebih jauh dari server utama. Ketika pengguna meminta video untuk diputar, CDN menyajikan video dari server terdekat berdasarkan lokasi geografis pengguna, mengurangi waktu tunggu untuk pengunduhan atau streaming.

1.2. Describe the overall workflow of loading, editing, and exporting a video project.

Pengunggahan Video (Loading Video)

1.2.1. Langkah : Pengguna Mengakses Aplikasi

1.2.1.1. Pengguna mengakses aplikasi video editor melalui browser mereka, yang terhubung dengan server backend. Antarmuka pengguna (frontend) menampilkan halaman utama dari editor video.

1.2.2. Langkah 2: Pengguna Mengunggah Video

1.2.2.1. Pengguna mengklik tombol unggah video pada antarmuka aplikasi.

1.2.2.2. Setelah pengguna memilih file video dari perangkat mereka, browser mengirimkan permintaan POST ke API server untuk mengunggah file tersebut.

1.2.3. Langkah 3: Pengolahan File oleh Backend

1.2.3.1. API server menerima file video yang diunggah dan memprosesnya. Biasanya, backend akan mengonversi atau memeriksa format video agar sesuai dengan sistem yang digunakan oleh aplikasi.

1.2.3.2. File Storage (Cloud) menyimpan file video yang diunggah untuk pengeditan lebih lanjut. File ini disimpan di tempat penyimpanan yang aman dan dapat diakses sesuai kebutuhan.

1.2.3.3. Database/Metadata mencatat informasi terkait file video, seperti nama file, durasi, format, resolusi, dan status pengunggahan.

1.2.4. Langkah 4: Menyimpan dan Menampilkan Video di Timeline

1.2.4.1. Setelah file video berhasil diunggah dan disimpan, metadata video (seperti thumbnail dan durasi) akan diambil oleh frontend untuk ditampilkan di Timeline Editor di browser.

1.2.4.2. Video yang diunggah kemudian diputar di Video Player yang ada di frontend untuk memungkinkan pengguna melihat pratinjau video.

Pengeditan Video (Editing the Video)

1.2.1. Langkah 1: Menyesuaikan Video di Timeline

1.2.1.1. Pengguna mulai mengedit video dengan menambahkan atau mengubah elemen dalam Timeline Editor.

1.2.1.2. Pengguna dapat melakukan tindakan seperti memotong, menambah efek, atau menggabungkan beberapa klip video.

1.2.1.3. Timeline Editor menampilkan video dalam bentuk urutan klip yang dapat dipindahkan atau dipotong oleh pengguna.

1.2.2. Langkah 2: Interaksi dengan Video Player

1.2.2.1. Video Player menampilkan video yang sedang diputar, memungkinkan pengguna untuk memonitor perubahan yang mereka buat di Timeline Editor secara langsung.

1.2.2.2. Pengguna dapat memilih bagian video yang ingin dipotong atau diberi efek tertentu dan memanipulasi elemen-elemen tersebut di Canvas yang terhubung dengan timeline.

1.2.3. Langkah 3: Pengiriman Permintaan ke Backend

1.2.3.1. Setiap perubahan yang dilakukan oleh pengguna di Timeline Editor (misalnya, menambah transisi atau efek) akan dikirim melalui permintaan API (biasanya POST atau PATCH) ke backend untuk diproses lebih lanjut.

1.2.3.2. Backend menerima data dari frontend yang berisi informasi tentang pengeditan yang diinginkan dan menerapkan perubahan tersebut pada file video yang ada.

1.2.4. Langkah 4: Pemrosesan Video oleh Backend

1.2.4.1. Video Processing Engine di backend memproses file video berdasarkan perubahan yang telah dilakukan di timeline.

Pemrosesan ini dapat meliputi pemotongan video, penerapan efek visual atau transisi, dan penggabungan klip video.

1.2.4.2. Pemrosesan ini biasanya menggunakan alat seperti FFmpeg untuk transcoding dan rendering video.

1.2.5. Langkah 5: Pembaruan Metadata dan Penyimpanan

1.2.5.1. Database/Metadata diperbarui dengan status terbaru dari pengeditan, termasuk informasi mengenai transisi yang diterapkan, durasi video yang telah dipotong, dan efek lainnya.

1.2.5.2. Setelah pengeditan selesai, video yang telah diproses disimpan kembali di File Storage untuk keperluan ekspor atau distribusi.

1.2.6. Langkah 6: Pratinjau Hasil Pengeditan

1.2.6.1. Setelah pemrosesan selesai, hasil video yang telah diedit dapat diputar kembali di Video Player di browser untuk melihat pratinjau hasil pengeditan.

1.2.6.2. Jika pengguna puas dengan hasilnya, mereka dapat melanjutkan ke tahap ekspor. Jika tidak, mereka dapat terus mengedit video di Timeline Editor.

Ekspor Video (Exporting the Video)

1.2.1. Langkah 1: Persiapan Ekspor

1.2.1.1. Setelah pengeditan selesai, pengguna dapat memilih opsi ekspor atau render untuk menghasilkan video final.

1.2.1.2. Frontend mengirimkan permintaan ekspor ke backend, yang berisi format video yang diinginkan (misalnya, MP4, MOV) dan resolusi akhir.

1.2.2. Langkah 2: Proses Rendering oleh Backend

1.2.2.1. Backend menerima permintaan ekspor dan meneruskan video ke Video Processing Engine untuk proses rendering akhir.

1.2.2.2. Proses rendering ini bisa memakan waktu, tergantung pada panjang video, efek yang diterapkan, dan kualitas video yang diinginkan.

1.2.2.3. Setelah rendering selesai, file video final disimpan di File Storage untuk diakses dan diunduh oleh pengguna.

1.2.3. Langkah 3: Penyimpanan Hasil Ekspor

1.2.3.1. Video yang telah dirender dan siap diekspor disimpan di File Storage untuk distribusi lebih lanjut. File ini bisa disalin ke CDN jika aplikasi menginginkan distribusi video ke pengguna lain.

1.2.3.2. Database/Metadata juga diperbarui untuk mencatat bahwa video telah selesai diproses dan siap diunduh.

1.2.4. Langkah 4: Pengguna Mendapatkan Hasil Ekspor

1.2.4.1. Setelah video selesai dirender, pengguna diberi pilihan untuk mengunduh video atau menyebarkan video tersebut melalui berbagai platform.

1.2.4.2. Jika pengguna memilih untuk mengunduh, video dapat diambil langsung dari File Storage melalui permintaan HTTP GET dan disajikan oleh CDN untuk mempercepat pengunduhan (terutama untuk pengguna yang berada jauh dari server utama).

1.2.5. Langkah 5: Pengunduhan atau Penyebaran

1.2.5.1. Video final diunduh ke perangkat pengguna atau dibagikan melalui CDN untuk distribusi lebih luas.

1.2.5.2. Jika video dibagikan, CDN akan memastikan video dapat diakses dengan cepat dari lokasi server yang paling dekat dengan pengguna akhir.

1.3. Identify core modules and functionality, considering features like: timeline, video previews, effects, transitions, audio editing, project management etc.

Timeline Editor

1.3.1. Fungsi:

1.3.1.1. Menyusun urutan klip video.

1.3.1.2. Memungkinkan pengguna untuk mengedit urutan video dengan cara drag-and-drop.

1.3.1.3. Memungkinkan pengguna untuk menambahkan dan mengedit elemen lain seperti teks, gambar, atau transisi antar klip.

1.3.1.4. Pengguna dapat melakukan pemotongan, penggabungan, atau pemindahan klip dalam timeline.

1.3.2. Komponen Terkait:

1.3.2.1. Frontend (Browser): Antarmuka pengguna untuk timeline editor. Pengguna dapat menggeser klip video sepanjang timeline, menambahkan elemen, dan memanipulasi pengeditan.

1.3.2.2. Canvas & Timeline Editor: Di dalam browser, canvas digunakan untuk menampilkan pratinjau visual dari video yang sedang diedit, sementara timeline editor memberikan pengontrolan yang lebih baik atas urutan video.

1.3.2.3. Backend: Backend menerima data tentang perubahan timeline yang dikirim oleh frontend dan memperbarui metadata atau file video sesuai dengan pengeditan yang dilakukan.

1.3.3. Interaksi:

1.3.3.1. Pengguna dapat menyeret klip video atau elemen ke timeline, memindahkannya ke posisi baru, atau menyesuaikan durasinya.

1.3.3.2. Setiap perubahan yang terjadi di timeline akan diperbarui di Database/Metadata untuk mencatat status dan informasi baru terkait proyek.

1.3.3.3. Video Processing Engine bertanggung jawab untuk memproses pengeditan ini saat ekspor dilakukan.

Pratinjau Video (Video Preview)

1.3.1. Fungsi:

1.3.1.1. Menyediakan tampilan langsung dari video yang sedang diedit untuk memberikan umpan balik visual kepada pengguna.

1.3.1.2. Memungkinkan pengguna untuk melihat klip video, transisi, efek, dan perubahan lainnya secara real-time sebelum mengekspor video akhir.

1.3.2. Komponen Terkait:

1.3.2.1. Video Player: Digunakan untuk menampilkan video yang sedang diedit di frontend. Video player ini menerima file video dari File Storage dan menampilkannya sesuai dengan pengeditan yang sedang dilakukan.

1.3.2.2. Frontend (Browser): Menampilkan antarmuka video player untuk memutar dan memantau perubahan yang dilakukan oleh pengguna.

1.3.2.3. Cache (Server-side & Client-side): Menyimpan sementara hasil pratinjau untuk mengurangi waktu tunggu dalam memuat pratinjau dan mempercepat respons.

1.3.3. Interaksi:

1.3.3.1. Frontend mengirimkan permintaan ke API Server untuk mengambil file video yang diperlukan, baik yang sudah diunggah atau hasil pengeditan terbaru.

1.3.3.2. Hasil pratinjau diputar langsung di Video Player, memungkinkan pengguna untuk melihat perubahan yang diterapkan dalam waktu nyata.

Efek Video (Effects)

1.3.1. Fungsi:

1.3.1.1. Menambahkan efek visual pada video, seperti filter warna, blur, perubahan kecepatan (slow motion/fast motion), atau efek khusus lainnya (misalnya, efek vintage, efek glitch).

1.3.1.2. Pengguna dapat menyesuaikan intensitas atau durasi efek sesuai kebutuhan.

1.3.2. Komponen Terkait:

1.3.2.1. Frontend (Browser): Menyediakan antarmuka bagi pengguna untuk memilih dan menerapkan efek pada klip video yang sedang diedit.

1.3.2.2. Canvas & Timeline Editor: Di dalam timeline, efek diterapkan pada klip video tertentu, yang memungkinkan pengguna melihat dan menyesuaikan efek secara langsung.

1.3.2.3. Video Processing Engine: Backend memproses dan menerapkan efek yang dipilih oleh pengguna pada video saat video sedang diproses.

1.3.3. Interaksi:

1.3.3.1. Pengguna memilih efek dari panel efek di Frontend, kemudian efek diterapkan langsung pada klip video yang ada di Timeline Editor.

1.3.3.2. Setiap perubahan pada efek dikirim ke Backend, di mana Video Processing Engine akan memanipulasi file video sesuai dengan efek yang diterapkan dan hasilnya akan disimpan di File Storage.

Transisi Video (Transitions)

1.3.1. Fungsi:

1.3.1.1. Menambahkan transisi antara dua klip video atau elemen, seperti efek fade in/fade out, slide, wipes, atau transisi khusus lainnya.

1.3.1.2. Memungkinkan pengguna untuk mengontrol durasi transisi dan jenis transisi yang diterapkan.

1.3.2. Komponen Terkait:

1.3.2.1. Frontend (Browser): Menyediakan antarmuka di mana pengguna dapat memilih dan menerapkan transisi antar klip dalam timeline.

1.3.2.2. Timeline Editor: Di sini pengguna dapat menempatkan dan menyesuaikan transisi antara klip video di timeline.

1.3.2.3. Video Processing Engine: Setelah pengguna selesai mengedit, transisi diterapkan dan diproses oleh backend untuk dimasukkan dalam video yang sudah selesai.

1.3.3. Interaksi:

1.3.3.1. Frontend mengirimkan data terkait transisi antar klip ke API Server.

1.3.3.2. Backend menerapkan transisi sesuai dengan pengaturan yang dibuat pengguna melalui Video Processing Engine, yang kemudian memodifikasi file video.

1.3.3.3. Hasil akhir video dengan transisi diterapkan disimpan kembali di File Storage untuk ekspor.

Pemotongan Video (Cut Video)

1.3.1. Fungsi:

1.3.1.1. Memotong Video: Modul ini memungkinkan pengguna untuk memotong bagian dari video yang tidak diinginkan atau untuk membagi video menjadi beberapa bagian dengan memilih titik masuk (in-point) dan titik keluar (out-point) di timeline.

1.3.1.2. Menyesuaikan Durasi: Pengguna dapat menentukan durasi setiap klip yang diinginkan dengan memotong video di bagian yang tidak diperlukan, atau hanya memilih bagian tertentu dari video untuk digunakan dalam proyek.

1.3.1.3. Pengaturan Waktu: Pengguna dapat memotong video dengan presisi berdasarkan detik atau frame untuk memastikan pengeditan yang sangat tepat.

1.3.2. Komponen Terkait:

1.3.2.1. Frontend (Browser): Di sini pengguna dapat mengatur titik awal dan akhir (in-point dan out-point) untuk memotong video menggunakan antarmuka timeline. Pengguna dapat menggunakan kontrol seperti slider untuk menyesuaikan durasi klip dan memvisualisasikan bagian yang ingin dipotong. Timeline Editor menyediakan antarmuka tempat pengguna dapat memilih bagian video untuk dipotong atau dipisahkan.

1.3.2.2. Timeline Editor (Canvas): Menampilkan video yang sedang diedit secara visual di dalam timeline. Bagian yang dipilih untuk dipotong akan ditampilkan dengan jelas. Ketika pengguna memilih titik in dan out, timeline editor memberikan umpan balik langsung mengenai bagian video yang dipotong dan bagian yang tetap.

1.3.2.3. Video Processing Engine: Setelah pemotongan dilakukan, Video Processing Engine di backend bertanggung jawab untuk memproses dan mengekspor bagian video yang dipilih. Engine ini akan memotong file video di server sesuai dengan bagian yang ditentukan oleh pengguna (in dan out points), lalu menyimpan file yang telah dipotong. Alat pemrosesan video digunakan untuk

melakukan pemotongan video sesuai dengan instruksi dari frontend.

1.3.2.4. File Storage: Setelah pemotongan selesai, bagian video yang telah dipotong disimpan di File Storage, baik itu Cloud Storage atau On-premise Storage. File Storage menyimpan video yang telah dipotong untuk digunakan lebih lanjut dalam proyek, atau untuk diekspor.

1.3.2.5. Database/Metadata: Metadata proyek diperbarui dengan informasi tentang video yang telah dipotong. Ini termasuk informasi seperti durasi baru video, titik awal dan akhir yang dipilih, serta file video yang dihasilkan setelah pemotongan.

1.3.3. Interaksi:

1.3.3.1. Frontend: Pengguna memilih bagian dari video yang ingin dipotong menggunakan kontrol yang ada di Timeline Editor. Pengguna menyesuaikan slider untuk menentukan titik in dan out yang diinginkan.

1.3.3.2. Backend: API Server menerima permintaan dari frontend dan meneruskan informasi titik pemotongan (in-point dan out-point) ke Video Processing Engine. Video Processing Engine kemudian melakukan pemotongan video menggunakan alat pemrosesan video seperti FFmpeg, di mana video asli akan dibagi sesuai dengan durasi yang dipilih, dan bagian yang tidak diinginkan akan dihapus. Hasil pemotongan video yang telah selesai diproses disimpan di File Storage untuk digunakan lebih lanjut dalam proyek.

1.3.3.3. Database/Metadata: Setelah pemotongan selesai, Database diperbarui untuk mencatat perubahan yang terjadi, termasuk durasi video baru dan file video yang telah dipotong.

Pengeditan Audio (Audio Editing)

1.3.1. Fungsi:

1.3.1.1. Memungkinkan pengguna untuk menambah, menghapus, dan mengedit trek audio dalam video.

1.3.1.2. Pengguna dapat mengubah volume, memotong bagian audio, atau menyinkronkan audio dengan klip video.

1.3.2. Komponen Terkait:

1.3.2.1. Frontend (Browser): Menyediakan antarmuka untuk mengedit trek audio, seperti menambah audio latar belakang, menyesuaikan volume, atau memotong bagian audio.

1.3.2.2. Audio Track: Bagian dari timeline di mana trek audio dapat ditambahkan, disesuaikan, atau dipotong sesuai dengan kebutuhan.

1.3.2.3. Video Processing Engine: Mengolah dan menerapkan perubahan pada audio, seperti pengeditan volume atau pencocokan waktu audio dengan video.

1.3.3. Interaksi:

1.3.3.1. Frontend mengirimkan permintaan untuk menambah atau mengubah trek audio ke API Server.

1.3.3.2. Backend memproses perubahan tersebut dan mengirimkan file audio yang sudah diedit untuk digabungkan dengan file video. Perubahan ini akan disimpan di File Storage setelah selesai.

Manajemen Proyek (Project Management)

1.3.1. Fungsi:

1.3.1.1. Mengelola proyek video yang sedang dikerjakan, termasuk menyimpan status, metadata, dan referensi ke file video dan audio terkait.

1.3.1.2. Memungkinkan pengguna untuk membuka, menyimpan, dan melanjutkan proyek video yang sedang dikerjakan.

1.3.2. Komponen Terkait:

1.3.2.1. Database/Metadata: Menyimpan informasi tentang proyek video, seperti nama proyek, status pengeditan, dan file video/audio yang digunakan dalam proyek.

1.3.2.2. Frontend (Browser): Memberikan antarmuka bagi pengguna untuk membuat, menyimpan, dan memuat proyek mereka.

1.3.2.3. File Storage: Menyimpan file video yang terkait dengan proyek, termasuk video yang sedang dikerjakan dan hasil ekspor.

1.3.3. Interaksi:

1.3.3.1. Frontend mengirimkan data terkait status proyek (seperti status pengeditan atau lokasi file) ke API Server.

1.3.3.2. Backend menyimpan data proyek di Database/Metadata dan mengelola file yang terkait dengan proyek di File Storage.

1.3.3.3. Pengguna dapat membuka proyek yang sudah ada atau menyimpan proyek baru yang sedang dikerjakan, memastikan bahwa semua perubahan disimpan dengan benar.

2. Front-end Architecture:

2.1. Describe how the front-end (UI) would be structured and which technologies you would pick to handle the complexity.

React dipilih untuk pengembangan antarmuka pengguna (UI). React adalah pustaka JavaScript yang dikembangkan oleh Facebook, yang dirancang untuk membangun antarmuka pengguna (UI) yang dinamis dan efisien dengan cara berbasis komponen. Salah satu alasan utama mengapa React digunakan dalam pengembangan aplikasi video editor berbasis web adalah kemampuannya untuk membuat aplikasi yang sangat responsif dan modular. React memungkinkan pengembang untuk membagi UI menjadi bagian-bagian kecil dan terpisah (komponen) yang dapat digunakan kembali, memudahkan pengelolaan kode dan pengembangan aplikasi yang kompleks. Selain itu, React menggunakan **virtual DOM**, yang secara signifikan meningkatkan kinerja aplikasi dengan meminimalkan pembaruan elemen-elemen UI yang tidak diperlukan, sehingga aplikasi tetap cepat meskipun menangani interaksi pengguna yang intensif, seperti pengeditan video dalam aplikasi ini. Dengan dukungan kuat dari ekosistem JavaScript, berbagai pustaka terkait, dan komunitas yang besar, React menjadi pilihan yang ideal untuk membangun aplikasi video editor berbasis web yang modern, efisien, dan mudah dipelihara.

Home Page (Informasi Umum Produk)

2.1.1. Header: Termasuk logo aplikasi, menu navigasi (Home, Edit Video, Hasil Video, Login/Daftar jika diperlukan)

2.1.2. Hero Section: Gambar atau video yang menarik perhatian yang memperkenalkan aplikasi dan menggambarkan cara kerja aplikasi (misalnya, "Mulai Edit Video Anda Sekarang").

2.1.3. Fitur Utama: Menampilkan fitur utama aplikasi, seperti timeline editor, efek, transisi, dan alat pengeditan audio.

2.1.4. Tombol Aksi: Tombol untuk mulai mengedit video atau mengunduh aplikasi jika diperlukan.

2.1.5. Testimoni/Review Pengguna: Bagian ini bisa menampilkan umpan balik atau testimoni pengguna yang telah menggunakan aplikasi.

Edit Video Page

2.1.1. Header: Menampilkan nama aplikasi dan tautan navigasi ke halaman lain (Home, Hasil Video).

2.1.2. Sidebar: Di sebelah kiri layar, dapat berisi alat-alat seperti panel efek, kontrol audio, serta menu untuk memilih atau menambahkan media, dll. Sidebar dapat menyembunyikan/menampilkan alat-alat sesuai kebutuhan.

2.1.3. Timeline Editor: Tempat utama di mana pengguna dapat menyeret dan melepaskan klip video, mengatur durasi, menambahkan efek dan transisi.

2.1.4. Video Player: Memutar video yang sedang diedit, memberikan umpan balik visual secara langsung kepada pengguna.

2.1.5. Tombol Aksi: Menyimpan proyek, melanjutkan pengeditan, atau mengekspor video.

2.1.6. Pratinjau: Bagian kecil untuk melihat pratinjau video setelah menerapkan perubahan.

Halaman Hasil Semua Video yang Sudah Pernah Diedit

2.1.1. Header: Menampilkan nama aplikasi, menu navigasi (Home, Edit Video, Hasil Video).

2.1.2. Daftar Video: Daftar video yang sudah diekspor, dilengkapi dengan thumbnail, nama video, dan durasi.

2.1.3. Pencarian dan Filter: Fitur pencarian untuk memudahkan menemukan video berdasarkan nama, tanggal, atau status.

2.1.4. Tombol Aksi: Setiap video memiliki tombol untuk mengedit ulang (kembali ke editor video) atau mengunduh.

2.1.5. Preview Video: Pengguna dapat melihat pratinjau kecil dari setiap video sebelum mengklik untuk detail lebih lanjut.

2.2. Explain how you would manage the state (data) of the video project in the front end.

Dalam aplikasi video editor berbasis React, manajemen state proyek video di frontend akan sangat krusial untuk memastikan data pengeditan dan status proyek tersimpan dengan baik dan dapat diakses oleh berbagai komponen secara efisien. Untuk itu, manajemen state dapat dibagi menjadi dua bagian utama: state lokal dan state global.

Untuk state lokal, menggunakan React Hooks, seperti useState untuk mengelola data yang bersifat lokal dalam komponen tertentu. Misalnya, dalam komponen VideoPlayer, kita dapat menggunakan useState untuk melacak status pemutaran video (play, pause) atau posisi video dalam timeline. Sedangkan untuk **state global**, yang mencakup data yang harus diakses oleh banyak komponen, seperti status pengeditan video (durasi klip, efek yang diterapkan, urutan klip, dll.), menggunakan Redux. **Redux** lebih cocok digunakan ketika aplikasi memiliki kompleksitas yang lebih tinggi dengan banyak state yang perlu dipertahankan dan dikelola. Dalam hal ini, Redux menyimpan semua data proyek di satu tempat (store) dan menyediakan cara untuk memperbarui state melalui dispatching action, yang akan mengubah state secara terpusat dan efisien.

2.3. Explain how you would manage the integration of the UI with the backend part. Optional, explain a bit about the backend system and how it handles the video editor features.

Untuk mengintegrasikan antarmuka pengguna (UI) dengan backend dalam aplikasi video editor berbasis web, komunikasi berbasis API seperti RESTful API digunakan untuk memungkinkan frontend (React) berinteraksi dengan backend guna mengambil dan mengirimkan data secara efisien. Antarmuka pengguna akan mengirimkan permintaan HTTP menggunakan pustaka seperti axios atau fetch untuk mengambil atau mengirimkan data terkait video, metadata, status proyek, atau hasil

pengeditan. Misalnya, ketika pengguna mengunggah video untuk diedit, frontend mengirimkan file video dan metadata terkait ke backend melalui metode POST. Backend kemudian akan menyimpan file video di File Storage (Google Cloud Storage), memperbarui metadata di Database, dan mengembalikan informasi status kepada frontend.

Di sisi backend, sistem dapat dibangun menggunakan teknologi seperti Node.js dengan Express untuk menangani permintaan HTTP, mengelola file video, dan menjalankan proses pengeditan menggunakan alat seperti FFmpeg. Backend bertanggung jawab untuk menangani pengeditan video yang intensif, seperti pemotongan klip, penerapan efek visual, atau rendering video. Proses ini biasanya dilakukan secara asinkron menggunakan job queues (misalnya RabbitMQ atau Bull), yang memungkinkan backend untuk memproses video tanpa memblokir permintaan lainnya. Setelah pemrosesan video selesai, backend menyimpan hasil video di File Storage, memperbarui Database dengan status terbaru, dan mengembalikan URL atau path video yang telah diproses ke frontend untuk ditampilkan atau diunduh oleh pengguna. Dengan pendekatan ini, backend bertanggung jawab atas proses pengeditan berat, sementara frontend memastikan interaksi yang lancar dan responsif bagi pengguna.

3. Performance Considerations:
   3.1. Discuss performance optimizations that can be made for video editing in the browser (e.g., rendering, caching, etc).

Untuk mengoptimalkan kinerja aplikasi video editor berbasis browser, beberapa strategi dapat diterapkan, terutama karena pengeditan video dapat menjadi proses yang sangat intensif baik dari sisi komputasi maupun penggunaan memori. Salah satu optimasi utama adalah rendering. Untuk meningkatkan kecepatan render video, teknik lazy loading dan virtualization dapat digunakan, terutama untuk elemen-elemen seperti timeline yang panjang atau daftar klip video. Dengan lazy loading, hanya bagian dari video atau timeline yang sedang terlihat di layar yang akan dimuat dan direnderi, mengurangi beban pada DOM dan meningkatkan responsivitas aplikasi. React.memo dan React.lazy juga dapat digunakan untuk menghindari render ulang komponen yang tidak berubah, memastikan bahwa hanya komponen yang benar-benar membutuhkan pembaruan yang direnderi ulang.

Selain itu, caching juga berperan penting dalam meningkatkan kinerja. Cache di sisi klien dapat digunakan untuk menyimpan data video yang sering diakses, seperti thumbnail video atau bagian dari video yang telah diproses, sehingga tidak perlu mengunduh atau memproses ulang data yang sama setiap kali. Teknik service workers dapat diterapkan untuk menyimpan file video dan data terkait secara lokal di browser, memungkinkan pengeditan dilakukan secara offline atau lebih cepat dengan mengakses data dari cache. Di sisi backend, content delivery network (CDN) dapat digunakan untuk mengirimkan video dan file terkait dengan lebih cepat ke pengguna berdasarkan lokasi geografis mereka, mengurangi latensi saat memuat video.

Multithreading dengan Web Workers dapat digunakan untuk memisahkan pemrosesan video berat ke thread terpisah, memastikan bahwa UI tetap responsif saat video sedang diproses. Terakhir, pengurangan resolusi video saat pratinjau atau pengeditan sementara, serta menggunakan teknik kompresi video yang efisien, dapat mengurangi waktu pemrosesan dan penggunaan sumber daya, memastikan pengalaman pengguna yang lebih cepat dan lancar meskipun dengan video berukuran besar. Dengan kombinasi optimasi ini, aplikasi video editor berbasis browser dapat memberikan performa yang lebih baik tanpa mengorbankan fungsionalitas atau kualitas.

3.2. Address how you would manage browser limitations regarding memory and processing.

3.2.1. Penggunaan Web Workers untuk Pemrosesan Asinkron

Browser memiliki keterbatasan dalam hal pemrosesan satu thread, sehingga aplikasi yang berat secara komputasi, seperti pengeditan video, dapat menyebabkan UI menjadi lambat atau tidak responsif. Untuk mengatasi hal ini, Web Workers dapat digunakan untuk memindahkan pemrosesan video berat ke thread terpisah. Web Workers memungkinkan eksekusi proses di luar thread utama, sehingga antarmuka pengguna tetap responsif, bahkan ketika video sedang diproses. Misalnya, saat memotong video atau menerapkan efek, proses tersebut dapat dijalankan dalam Web Worker terpisah, sehingga pengguna masih dapat berinteraksi dengan aplikasi tanpa gangguan.

3.2.2. Optimasi Penggunaan Memori dengan Teknik Lazy Loading dan Virtualisasi

Untuk mengatasi keterbatasan memori, terutama ketika bekerja dengan banyak data video atau timeline yang panjang, teknik lazy loading dan virtualisasi dapat diterapkan. Lazy loading hanya memuat dan merender bagian data yang diperlukan pada saat itu, bukan memuat seluruh konten sekaligus. Dalam konteks editor video, ini berarti hanya bagian dari timeline atau video yang terlihat yang akan dimuat dan dirender, mengurangi beban memori. React Virtualization (seperti menggunakan pustaka react-window) juga memungkinkan merender hanya elemen-elemen yang terlihat di layar, yang membantu mengurangi jumlah elemen DOM yang harus dikelola oleh browser.

3.2.3. Penggunaan Kompresi Video dan Pengurangan Resolusi Sementara

Keterbatasan memori juga dapat diatasi dengan mengurangi resolusi video sementara selama pengeditan. Video dengan resolusi tinggi memerlukan lebih banyak memori dan waktu pemrosesan, jadi saat melakukan pratinjau atau pengeditan, video dapat dikompresi atau diturunkan resolusinya untuk mengurangi beban pada memori dan CPU. Setelah pengeditan selesai, video dapat diekspor dalam resolusi aslinya. Teknik kompresi video yang efisien juga dapat digunakan untuk mengurangi ukuran file video saat diunggah atau diekspor, memastikan bahwa aplikasi tidak kehabisan memori atau terhambat oleh pengolahan file besar.

3.2.4. Pengelolaan Cache di Browser

Untuk mengurangi beban pemrosesan berulang, cache di sisi klien dapat digunakan untuk menyimpan data sementara, seperti thumbnail video, potongan klip yang sering digunakan, atau metadata yang sudah diproses. Penggunaan service workers memungkinkan cache ini dikelola secara efisien, memastikan bahwa aplikasi dapat memuat data dari cache daripada memproses ulang setiap kali pengguna melakukan perubahan pada video. Ini sangat membantu mengurangi penggunaan memori dan waktu pemrosesan saat pengguna melakukan editing berulang-ulang.

3.2.5. Pembatasan Jumlah Data yang Dimuat

Untuk mengatasi keterbatasan memori di browser, aplikasi video editor dapat membatasi jumlah data video yang dimuat dalam satu waktu. Sebagai contoh, alih-alih memuat seluruh video atau proyek dalam satu kali, hanya

segmen-segmen kecil yang akan dimuat untuk pengeditan, sementara sisanya diakses sesuai kebutuhan. Ini dapat diatur dengan mengatur batasan durasi klip yang dimuat di timeline dan hanya memproses bagian-bagian yang sedang diedit atau dipilih pengguna.

3.2.6. Pemanfaatan Cloud Computing dan Offloading

Aplikasi video editor berbasis browser dapat memanfaatkan cloud computing untuk memindahkan sebagian pemrosesan berat ke server cloud, mengurangi beban pada perangkat pengguna. Pengolahan video seperti rendering, transcoding, atau penerapan efek dapat dilakukan di server cloud, dan hasilnya dikirim kembali ke browser untuk ditampilkan kepada pengguna. Dengan offloading pemrosesan berat ke server cloud, penggunaan memori dan CPU di perangkat pengguna dapat diminimalkan.

3.2.7. Monitoring Penggunaan Memori

Sangat penting untuk terus memantau penggunaan memori aplikasi selama pengeditan. Browser menyediakan alat untuk memonitor penggunaan memori melalui devtools yang memungkinkan pengembang untuk melihat dan mengoptimalkan penggunaan memori dalam aplikasi. Ini memungkinkan pengembang untuk mendeteksi kebocoran memori dan bagian aplikasi yang terlalu berat atau memerlukan pengelolaan sumber daya yang lebih baik.

4. Scalability:

4.1. How would the editor handle larger and complex projects.

Untuk menangani proyek video yang lebih besar dan kompleks, aplikasi editor video berbasis browser perlu mengimplementasikan arsitektur yang efisien dan scalable, serta teknik optimasi untuk menjaga performa dan keandalan. Salah satu pendekatan utama adalah dengan menggunakan arsitektur modular, di mana aplikasi dibagi menjadi komponen-komponen kecil yang dapat digunakan kembali. Dengan cara ini, pengelolaan data dan rendering bisa dilakukan lebih efisien. Komponen seperti Timeline Editor, Video Player, Panel Efek, dan Audio Editor dapat beroperasi secara independen namun tetap saling terintegrasi. Untuk proyek besar, aplikasi juga harus mendukung lazy loading untuk memuat hanya bagian yang diperlukan dari timeline atau video, mengurangi beban memori dan meningkatkan responsivitas. Web Workers dapat digunakan untuk memindahkan pemrosesan video berat ke

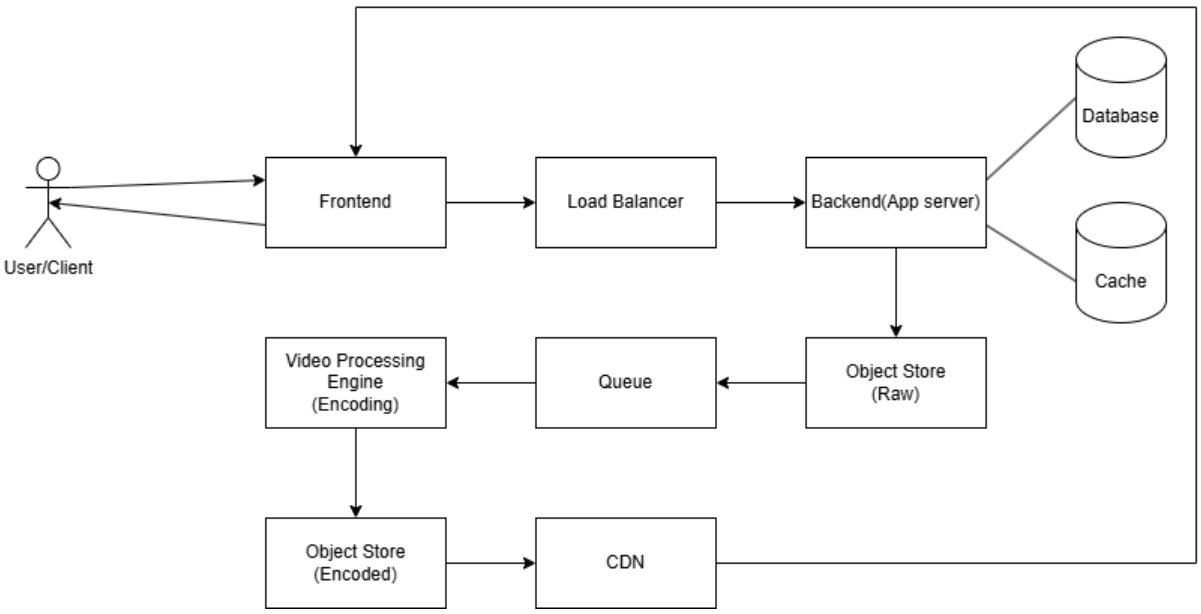
thread terpisah, sehingga UI tetap responsif meskipun proyek video yang diedit memiliki banyak klip atau efek. Selain itu, cloud computing dapat digunakan untuk memindahkan sebagian besar pemrosesan video (seperti rendering dan transcoding) ke server, mengurangi beban pada perangkat pengguna dan memungkinkan proyek yang lebih besar diproses dengan lebih cepat. Penggunaan CDN (Content Delivery Network) juga bisa mempercepat pengiriman video ke pengguna di berbagai lokasi, mengurangi latensi. Dengan penerapan teknik-teknik ini, editor video dapat menangani proyek yang lebih besar dan kompleks dengan tetap mempertahankan kinerja yang baik dan pengalaman pengguna yang lancar.

4.2. What consideration to take into account for future features.

Untuk fitur-fitur di masa depan, ada beberapa pertimbangan penting yang perlu diperhatikan dalam pengembangan aplikasi video editor berbasis browser. Pertama, skalabilitas sistem harus menjadi prioritas utama, mengingat proyek video yang lebih besar dan lebih kompleks akan membutuhkan lebih banyak sumber daya. Oleh karena itu, pengembangan harus mempertimbangkan penerapan cloud computing untuk pemrosesan yang lebih efisien dan skalabilitas yang lebih baik, serta penggunaan CDN untuk distribusi video yang lebih cepat. Selain itu, perlu ada pertimbangan terkait kompatibilitas perangkat karena semakin banyak pengguna yang menggunakan berbagai jenis perangkat dengan spesifikasi yang berbeda. Aplikasi harus dapat berjalan dengan baik di perangkat dengan kapasitas memori dan prosesor yang lebih rendah. Kemudian, pengeditan video kolaboratif juga bisa menjadi fitur penting di masa depan, sehingga perlu dipikirkan penggunaan WebSockets atau teknologi serupa untuk memungkinkan banyak pengguna berkolaborasi secara real-time dalam proyek yang sama. Fitur tambahan seperti penerapan AI dan machine learning untuk otomatisasi beberapa tugas pengeditan video, seperti pemilihan klip terbaik atau penerapan efek secara otomatis, juga perlu dipertimbangkan untuk meningkatkan efisiensi pengeditan. Selain itu, untuk mengakomodasi perkembangan teknologi, aplikasi harus dapat dengan mudah diintegrasikan dengan format video baru dan teknologi streaming terbaru. Semua pertimbangan ini penting untuk memastikan bahwa aplikasi video editor tetap relevan, efisien, dan dapat memenuhi kebutuhan pengguna di masa depan.

5. Deliverables:

5.1.1. A document (text, diagram, or a combination) outlining the architecture.



5.1.1.1. Frontend (Sisi Klien)

Fungsi: Ini adalah antarmuka utama yang digunakan oleh pengguna untuk mengakses aplikasi editor video berbasis web. Pengguna dapat mengunggah video, mengeditnya, dan melihat hasil pengeditan melalui browser tanpa perlu menginstal perangkat lunak tambahan.

5.1.1.2. Load Balancer

Fungsi: Load balancer mendistribusikan trafik dan permintaan API ke berbagai server backend yang tersedia. Tujuannya untuk mencegah satu server menjadi terlalu terbebani dan memastikan skalabilitas aplikasi. Load balancer juga menjaga ketersediaan tinggi dan mengoptimalkan penggunaan sumber daya server.

5.1.1.3. Backend (App Server)

Fungsi: Server backend bertanggung jawab untuk menerima dan memproses permintaan dari frontend, seperti permintaan untuk mengambil metadata video, status pengeditan, atau hasil pemrosesan video. Backend juga mengelola logika aplikasi dan alur data antar komponen lain, seperti video processing engine dan database.

5.1.1.4. Database

Fungsi: Database menyimpan metadata terkait video (seperti nama file, durasi, format, dan status pengeditan) dan informasi pengguna. Semua data penting yang berhubungan dengan video dan proyek pengguna disimpan di sini.

5.1.1.5. Cache

Fungsi: Cache digunakan untuk menyimpan data yang sering diakses seperti metadata video atau thumbnail untuk mempercepat respons dan mengurangi waktu yang diperlukan untuk mengambil data dari server atau database.

5.1.1.6. Object Store (Raw)

Fungsi: Tempat penyimpanan untuk file video yang belum diproses (raw files). Ini termasuk video yang diunggah oleh pengguna sebelum diproses lebih lanjut.

5.1.1.7. Queue

Fungsi: Antrian pekerjaan digunakan untuk memproses tugas yang memakan waktu lama, seperti rendering atau transcoding video. Permintaan dari frontend akan dimasukkan ke dalam antrian dan diproses secara berurutan oleh worker. Ini memungkinkan backend menangani pemrosesan berat tanpa mengganggu pengalaman pengguna secara langsung.

5.1.1.8. Video Processing Engine (Encoding)

Fungsi: Komponen ini menangani tugas-tugas pemrosesan video berat, seperti transcoding, rendering, dan encoding video. Ini memungkinkan backend untuk mengedit dan mengonversi video sesuai dengan pengeditan yang dilakukan pengguna di frontend.

5.1.1.9. Object Store (Encoded)

Fungsi: Penyimpanan untuk file video yang telah diproses dan di-encode (hasil video yang telah diedit). Ini adalah tempat di mana video final yang sudah diproses disimpan untuk didistribusikan atau diunduh oleh pengguna.

5.1.1.10. Content Delivery Network (CDN)

Fungsi: CDN digunakan untuk mendistribusikan video yang telah diproses ke berbagai lokasi server global. Ini mengurangi latensi dan mempercepat waktu akses video bagi pengguna yang berada di lokasi geografis yang berbeda, meningkatkan pengalaman pengguna secara keseluruhan.

5.1.2. A rationale behind design decisions.

Keputusan desain arsitektur aplikasi video editor berbasis web ini didasarkan pada beberapa pertimbangan utama untuk memastikan kinerja yang optimal dan pengalaman pengguna yang lancar. Penggunaan frontend (browser) sebagai antarmuka pengguna memungkinkan akses yang mudah tanpa perlu perangkat lunak tambahan, sehingga pengguna dapat mengedit video dari berbagai perangkat. Untuk menangani lonjakan trafik dan memastikan aplikasi tetap responsif, load balancer digunakan untuk mendistribusikan trafik secara merata ke beberapa server backend. Hal ini juga mendukung high availability dan scalability, yang memungkinkan aplikasi untuk menangani volume pengguna yang besar dengan performa yang optimal.

Di sisi backend, peran app server sangat penting sebagai penghubung antara frontend dan berbagai komponen lain seperti video processing engine dan penyimpanan data di object store. Tanpa backend yang terstruktur, komunikasi antara frontend dan komponen lainnya akan menjadi tidak efisien. Backend juga mengelola autentikasi dan otorisasi pengguna, memastikan keamanan dan akses yang tepat. Untuk menangani proses pengeditan video yang intensif, video processing engine dipisahkan sebagai komponen terpisah. Ini memungkinkan aplikasi untuk mengelola tugas berat seperti transcoding dan rendering video tanpa membebani server backend atau frontend. Penggunaan teknologi seperti FFmpeg dalam video processing engine mempercepat pemrosesan dan meningkatkan pengalaman pengguna.

Selain itu, queue digunakan untuk memproses tugas yang memerlukan waktu lama secara asinkron, seperti transcoding atau rendering video, memastikan aplikasi tetap responsif dan tidak terhambat oleh proses berat. Dengan menggunakan antrian pekerjaan, tugas video dapat diproses secara berurutan tanpa saling mengganggu. Penyimpanan file video yang diunggah dan yang telah diproses dilakukan di object store berbasis cloud, yang terpisah untuk raw files dan encoded files. Penggunaan penyimpanan cloud memungkinkan aplikasi untuk menyimpan video dalam format yang lebih terstruktur dan siap untuk distribusi atau pengunduhan, serta memberikan skalabilitas dan fleksibilitas dalam mengelola volume data yang besar. Penyimpanan cloud juga mengurangi beban pada sistem penyimpanan utama, meningkatkan keandalan, dan memastikan akses yang cepat ke file video. Keputusan-keputusan ini diambil untuk menciptakan

arsitektur yang skalabel, efisien, dan mampu menangani beban tinggi dengan pengalaman pengguna yang optimal.