Data/Application Flow of YouTube Playlist Length Calculator

# 1. Playlist URL Submission Flow

## User Input:

• User enters YouTube playlist URL in PlaylistInput component  
• Component state tracks the input value via controlled input pattern

## Form Submission:

• User clicks 'Load Playlist' button, triggering the onSubmit handler  
• handleSubmit function validates the URL and prevents form default behavior  
• Valid URL triggers onSubmit callback passed from parent component

## State Update in App Component:

• handlePlaylistSubmit function in App.tsx receives the playlist URL  
• setPlaylistId updates the playlistId state with the new value

## Effect Trigger:

• useEffect hook with [playlistId] dependency triggers  
• Loading state is set to true via setIsLoading(true)  
• Previous data and errors are cleared

## API Request Preparation:

• Frontend prepares API calls using the api.ts utility functions  
• URL is processed to determine if it's a full URL or just an ID

## Backend API Call:

• Request is sent to /api/playlist/{playlist\_id} or /api/playlist-url endpoint  
• FastAPI route handler receives and validates the request  
• youtube\_service.get\_playlist\_info() method is called

## YouTube API Interaction:

• Backend makes HTTP requests to YouTube Data API  
• \_make\_request method handles communication with YouTube  
• Response data is processed and validated against Pydantic models

## Backend Response Formation:

• Data from YouTube API is transformed into application-specific models  
• Additional calculations (like estimated duration) are performed  
• Response object is created conforming to defined models

## Frontend Data Reception:

• API response is received in the frontend  
• Data is logged and formatted to match TypeScript interfaces  
• State updates are triggered for multiple state variables:  
 - setPlaylistInfo(formattedPlaylistInfo)  
 - setVideos(firstPageVideos)  
 - setNextPageToken(token)

## UI Update:

• React processes state changes and re-renders components  
• Loading state is set to false via setIsLoading(false)  
• PlaylistViewer receives new props and renders playlist data

# 2. Video Loading and Pagination Flow

## Initial Videos Load:

• First batch of videos (up to 20) loads with initial playlist data  
• Videos are stored in the videos state array  
• nextPageToken is stored for pagination

## Load More Interaction:

• User clicks 'Load More' button at the bottom of the video list  
• handleLoadMore function is triggered  
• Loading more state is set via setIsLoadingMore(true)

## Additional Videos Request:

• API call is made to /api/playlist/{playlist\_id}/videos with page\_token  
• Backend processes the request and fetches the next batch from YouTube  
• Response includes videos and a new nextPageToken (if more videos exist)

## State Update:

• New videos are appended to existing videos state:  
 - setVideos(prevVideos => [...prevVideos, ...moreVideos])  
• Next page token is updated for potential future requests  
• Loading state is reset

## UI Update:

• Component re-renders with the expanded video list  
• 'Load More' button remains if more videos are available (nextPageToken exists)

# 3. Playback Speed Adjustment Flow

## User Interaction:

• User adjusts the playback speed slider  
• onChange event triggers handleChange function in SpeedSlider  
• New speed value is passed to the parent via onChange callback

## State Update:

• handlePlaybackSpeedChange in App.tsx updates the state:  
 - setPlaybackSpeed(speed)

## Calculation Update:

• Inside PlaylistViewer, adjusted duration is calculated:  
 - const adjustedDurationSeconds = calculateAdjustedDuration(totalDurationSeconds, playbackSpeed)

## UI Update:

• Component re-renders with the new adjusted duration  
• Both original and adjusted durations are displayed