Architecture Design(ADD)

PlayWave: Video Streaming WebApp Remaining

Revision Number: 1.0
Last date of revision: 2025/02/04
Written By: Bishal Shahi

Document Version Control

Change Record:

Date Issued	Version	Description	Author
04/02/2025	1.0	Document Initialized	Bishal Shahi

Reviews:

Date Issued	Version	Description	Author
04/02/2025	1.0	Document Content	Bishal Shahi

Approval Status:

Date Issued	Version	Reviewed By	Approved By

Contents

1. Introduction

1.1 Why this Architecture Design Document?

The purpose of this **Architecture Design Document** is to provide a detailed breakdown of the **PlayWave Video Streaming App** architecture, ensuring a scalable, efficient, and maintainable system.

This document will:

- Define the **technical architecture** and its constraints.
- Provide an overview of **logging, error handling, and file storage**.
- Explain **deployment strategies and project flow**.

1.2 Scope

The Architecture Design focuses on defining **backend and frontend interactions**, **data flow**, **error handling**, and **deployment strategies using Vercel**.

1.3 Constraints

- **Local Storage**: The system does not use cloud storage solutions or databases.
- **FFmpeg Processing**: Transcoding videos locally may require optimized resource management.
- **No Authentication**: The system does not include user authentication mechanisms.

1.4 Risk

- **Performance Bottlenecks**: FFmpeg processing can slow down if multiple videos are uploaded.
- **Security Issues**: Open access APIs could be exploited.
- **Storage Overload**: Large video files can consume significant disk space.

2. Technical Specifications

2.1 Logging

- **Console Logging**: Basic logs for API requests and responses.
- **Error Logs**: Logged into a file for debugging purposes.

2.2 Error Handling

- **Try-Catch Blocks**: Used in API routes.
- **Centralized Error Middleware**: Manages unexpected errors.

2.3 File Storage

- **Multer**: Used to handle file uploads.
- **Local Directory**: Videos stored in a designated server folder.

2.4 Deployment

- **Vercel**: Used for deploying the frontend and backend.
- **Node.js Server**: Serves API and video files.

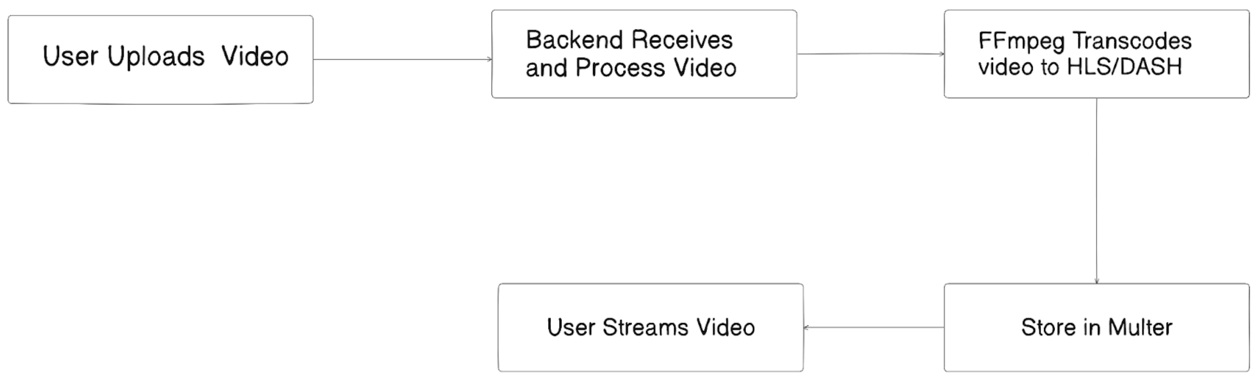
3. Technology Stack

Component	**Technology used**
Frontend	React, Tailwind css or Module css
Backend	Express (Node.js)
Video Processing	FFmpeg
Deployment	Vercel or Render

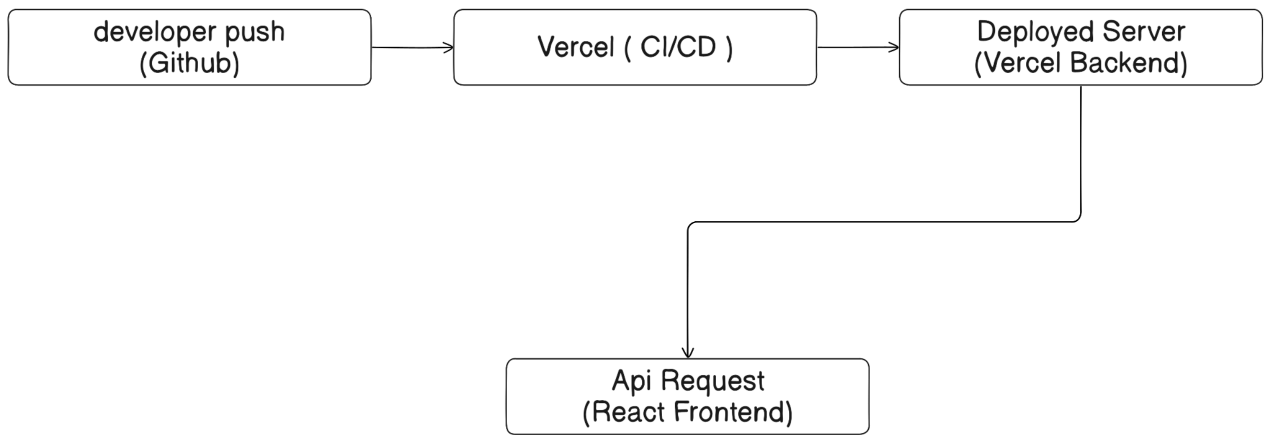
4. Proposed Solution

The **PlayWave** system follows a **file-based approach** where uploaded videos are stored locally and processed using FFmpeg before being served via API. This is made for experimental and learning purposes. With this learning and updating the extra features on this will make this video streamer a game changer having enhancements like less buffering, pure quality based on devices used.

5. Project Flow Diagram


User Uploads Video
Backend Receives and Process Video
FFmpeg Transcodes video to HLS/DASH
Store in Multer
User Streams Video

6. Deployment Flow Diagram


developer push (Github)
Vercel (CI/CD)
Deployed Server (Vercel Backend)
Api Request (React Frontend)

7. Testing and Evaluation

- **Unit Tests:** Jest for backend testing.
- **Manual Testing:** Ensuring smooth video upload and streaming.

8. Performance

- **Optimized FFmpeg Commands:** Reduces processing time.
- **Efficient API Calls:** Reduces latency.
- **Direct Vercel Deployment:** Simplifies hosting.