# Istoricul și Planul Strategic Final: Proiect "Sparky's Workshop Factory"

**Document de Referință Unic** **Ultima Versiune Integrată:** 18.3 (28 iunie 2025)

## Partea I-XI

*(Neschimbate)*

## Partea XII: Specificații de Produs pentru Versiunea 2.0 (Producție Multimedia)

*(Neschimbată)*

## NOU - Partea XIII: Plan de Implementare Tehnică pentru Modulul Video

**Obiectiv:** Integrarea logicii din notebook-ul public "Animated Story Video Generation" în backend-ul nostru pentru a alimenta componenta 🎬 Poveste Animată.

### 13.1. Pasul 1: Localizarea și Analiza Codului Sursă ("Gist-ului")

* **Acțiune:** Prima sarcină este găsirea notebook-ului public. O căutare Google pentru **"Gemini API Animated Story Video Generation Colab"** va duce cel mai probabil la resursa oficială de pe GitHub sau Google Codelabs.
* **Analiză:** După deschiderea notebook-ului, dezvoltatorul trebuie să identifice și să extragă următoarele elemente cheie:
  1. **Dependințele:** Bibliotecile Python necesare. Din imagini, acestea sunt cel puțin: google-generativeai, moviepy, asyncio.
  2. **Logica de Generare a Scenariului:** Celula de cod care construiește promptul pentru Gemini și parsează răspunsul JSON structurat.
  3. **Logica de Generare a Imaginilor:** Funcția sau celula care iterează prin scene și apelează API-ul Imagen.
  4. **Logica de Generare Audio:** Funcția care apelează API-ul Text-to-Speech (TTS).
  5. **Logica de Asamblare Video:** Partea de cod care folosește MoviePy pentru a combina imaginile și sunetul în clipuri și apoi pentru a le concatena.

### 13.2. Pasul 2: Adaptarea Codului din Notebook pentru un Backend de Producție

Codul dintr-un notebook rulează secvențial. Pentru serverul nostru, trebuie să-l refactorizăm în funcții modulare și robuste.

**Instrucțiuni de Implementare (în ai\_services.py):**

1. **Refactorizați Generarea Scenariului:**
   * **Creați o funcție:** generate\_structured\_story(theme: str, num\_scenes: int) -> dict:
   * **Logic:** Această funcție va prelua logica din prima secțiune a notebook-ului. Va construi promptul complex pentru Gemini, va apela API-ul și va returna obiectul JSON parsat care conține toate scenele.
2. **Refactorizați Generarea Media:**
   * **Creați o funcție:** generate\_scene\_assets(scene\_data: dict) -> dict:
   * **Logic:** Această funcție va primi datele pentru **o singură** scenă. Va face două apeluri în paralel:
     + Unul la generate\_image() cu image\_prompt-ul scenei.
     + Unul la un nou generate\_audio() cu narration\_text-ul scenei.
   * Va returna un dicționar cu URL-urile către imaginea și fișierul audio stocate pe Cloud Storage (ex: {'image\_url': '...', 'audio\_url': '...'}).
3. **Refactorizați Asamblarea Video:**
   * **Creați o funcție:** assemble\_video\_from\_assets(assets\_list: list, workshop\_id: str) -> str:
   * **Logic:** Această funcție va prelua logica de MoviePy. Va primi o listă de dicționare (rezultatul de la pasul anterior pentru toate scenele).
   * **Proces:**
     1. Va descărca temporar toate imaginile și fișierele audio de la URL-urile primite.
     2. Va folosi MoviePy pentru a crea un ImageClip și un AudioFileClip pentru fiecare scenă.
     3. Va combina clipurile și le va concatena.
     4. Va exporta fișierul .MP4 final.
     5. Va încărca MP4-ul final pe Cloud Storage.
     6. Va returna URL-ul public al videoclipului final.

### 13.3. Pasul 3: Integrarea în Funcția de Orchestrare a Backend-ului

Funcția principală de pe server, de exemplu handle\_animated\_story\_request(request\_data), va orchestra acum aceste funcții adaptate.

**Exemplu de Flux de Orchestrare:**

def handle\_animated\_story\_request(request\_data):  
 # Pasul 1: Generează scenariul  
 story\_json = ai\_services.generate\_structured\_story(  
 theme=request\_data['theme'],   
 num\_scenes=request\_data['num\_scenes']  
 )  
  
 # Pasul 2: Generează media pentru fiecare scenă  
 all\_assets = []  
 for scene in story\_json['scenes']:  
 scene\_assets = ai\_services.generate\_scene\_assets(scene)  
 all\_assets.append(scene\_assets)  
  
 # Pasul 3: Asamblează videoclipul final  
 final\_video\_url = ai\_services.assemble\_video\_from\_assets(  
 assets\_list=all\_assets,  
 workshop\_id=request\_data['workshop\_id']  
 )  
  
 # Pasul 4: Salvează în baza de date și returnează URL-ul  
 db.save\_video\_url(workshop\_id, final\_video\_url)  
 return {"videoUrl": final\_video\_url}

### 13.4. Managementul Dependințelor

* **Acțiune:** Adăugați bibliotecile necesare în fișierul requirements.txt al serviciului nostru backend.  
  google-generativeai  
  moviepy  
  # Alte dependențe necesare, precum google-cloud-storage

Prin urmarea acestor pași, putem prelua în mod eficient și profesional logica unui "gist" public, testat, și o putem integra ca un motor puternic și fiabil în inima platformei noastre.