FRONT-END SPEEL

FRONT-END SPEEL.CO

🎨 ESTRUCTURA GENERAL DE LA APLICACIÓN

Navegación Principal (Top Bar)

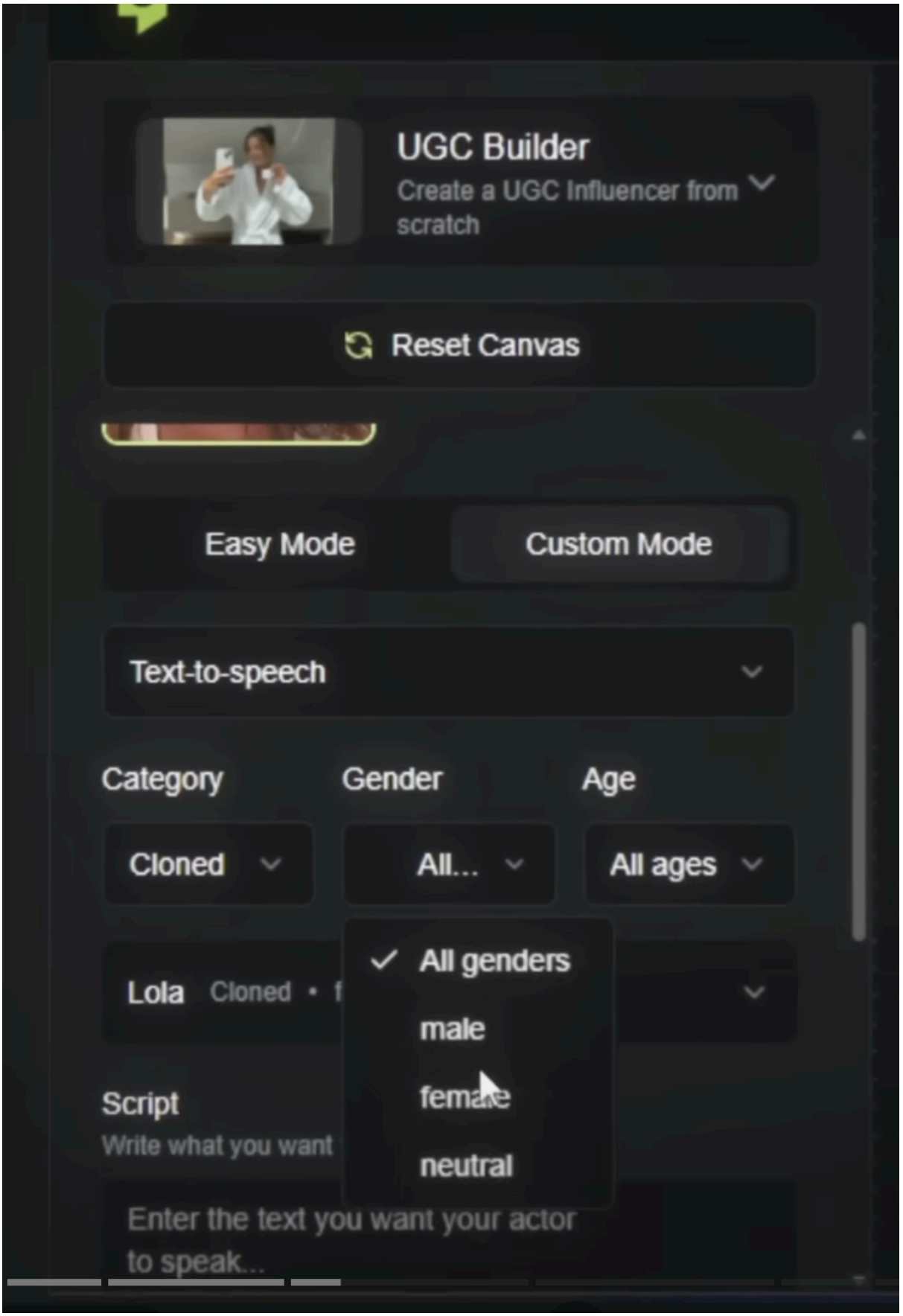
```
None
```

```
┌──────────────────────────────────────────────────────────┐
| [Logo Speel]  [UGC Builder] [Magic Edit] [Scenes]  |
|               [Images] [Videos] [Account]           |
└──────────────────────────────────────────────────────────┘
```

Tabs principales:

1. **UGC Builder** - Generación de videos
2. **Magic Edit** - Edición de imágenes (Nano Banana)
3. **Scenes and B-roll** - Generación con Sora 2
4. **Images** - Biblioteca de imágenes generadas
5. **Videos** - Biblioteca de videos generados
6. **AI Avatar Generator** - Creación de actores desde cero

📐 1. UGC BUILDER - INTERFAZ PRINCIPAL



Layout de 2 columnas:

```
None
┌──────────────────────┬──────────────────────────────────────┐
│   LEFT PANEL         │      RIGHT PANEL                     │
│   (Inputs)           │      (Preview/Output)                │
│                      │                                      │
│  [Mode Toggle]       │   [Video Preview Player]             │
│  [Image Upload]      │   [Loading State]                    │
│  [Script Input]      │   [Download Button]                  │
│  [Voice Select]      │   [Variations Grid]                  │
│  [Action Input]      │                                      │
│  [Generate BTN]      │                                      │
└──────────────────────┴──────────────────────────────────────┘
```

A. MODE TOGGLE (Componente Switch)

Ubicación: Top del panel izquierdo

```
None
<ModeToggle>
  <Option value="easy" active>Easy Mode</Option>
  <Option value="custom">Custom Mode</Option>
</ModeToggle>
```

Comportamiento:

- Toggle visual estilo iOS
- Cambia dinámicamente los campos disponibles
- Easy Mode: menos opciones, más automatizado
- Custom Mode: control total de voz y audio

Estados visuales:

- Active: Color primario (azul/morado)
- Inactive: Gris
- Hover: Ligero highlight

B. IMAGE UPLOAD SECTION

Componente: Drag & Drop + File Picker

```
None
<ImageUploadZone>
  {hasImage ? (
    <ImagePreview>
      <img src={uploadedImage} />
      <EditButton onClick={openEditor}>Edit Image</EditButton>
      <RemoveButton>×</RemoveButton>
    </ImagePreview>
  ) : (
    <DropZone>
      <Icon>📁</Icon>
      <Text>Upload image or drag here</Text>
      <Button>Browse Files</Button>
    </DropZone>
  )}
</ImageUploadZone>
```

Features técnicas:

- Soporte drag & drop
- Formatos: JPG, PNG, WEBP
- Preview instantáneo
- Botón "Edit Image" que abre Magic Edit en modal o nueva tab
- Botón "Remove" para limpiar
- Indicador de resolución/tamaño recomendado

Variante adicional: Selección desde biblioteca

```
None
<TabSwitch>
  <Tab>Upload</Tab>
  <Tab>Actor Library</Tab>
  <Tab>My Actors</Tab>
</TabSwitch>
```

C. SCRIPT INPUT (Diferente por modo)

EASY MODE:

```
None
<ScriptInput mode="easy">
  <Label>Script</Label>
  <Textarea
    placeholder="Enter what you want your actor to say..."
    rows={6}
    maxLength={500}
  />
  <CharacterCounter>0/500</CharacterCounter>
  <AIButton onClick={openScriptGPT}>
    ✨ Generate with AI
  </AIButton>
</ScriptInput>
```

Características:

- Textarea expandible
- Contador de caracteres en tiempo real
- Límite visual (cambio de color al acercarse al máximo)
- Botón de IA para abrir GPT helper (modal o link externo)
- Auto-save en localStorage cada 3 segundos

CUSTOM MODE:

```
None
<AudioSourceSelector>
  <TabGroup>
    <Tab active>Text to Speech</Tab>
    <Tab>Upload Audio</Tab>
    <Tab>Voice Changer</Tab>
  </TabGroup>

  {selectedTab === 'text-to-speech' && (
    <TextToSpeechPanel>
      <VoiceSelector>
        <Label>Select Voice</Label>
        <Dropdown>
          <Search placeholder="Search voices..." />
          <Filters>
```

```
            <Filter>Language</Filter>
            <Filter>Gender</Filter>
            <Filter>Age</Filter>
            <Filter>Accent</Filter>
          </Filters>
          <VoiceList>
            {voices.map(voice => (
              <VoiceItem>
                <Avatar src={voice.avatar} />
                <Name>{voice.name}</Name>
                <PlayButton onClick={() =>
playPreview(voice)}>▶</PlayButton>
              </VoiceItem>
            ))}
          </VoiceList>
        </Dropdown>
      </VoiceSelector>

      <ScriptTextarea
        placeholder="Enter your script here..."
      />

      <EmotionTags>
        <Label>Add Emotions (Optional)</Label>
        <TagInput
          placeholder="e.g., [happy], [sigh], [excited]"
        />
        <SuggestedTags>
          <Tag>[happy]</Tag>
          <Tag>[sad]</Tag>
          <Tag>[excited]</Tag>
          <Tag>[sigh]</Tag>
          <Tag>[curious]</Tag>
        </SuggestedTags>
      </EmotionTags>
    </TextToSpeechPanel>
  )}

  {selectedTab === 'upload-audio' && (
```

```
    <AudioUpload>
      <DropZone accept=".mp3,.wav,.m4a">
        <Icon>🎵</Icon>
        <Text>Upload audio file</Text>
      </DropZone>
      {audioFile && (
        <AudioPlayer src={audioFile} />
      )}
    </AudioUpload>
  )}

  {selectedTab === 'voice-changer' && (
    <VoiceChanger>
      <RecordButton>🎤 Record Your Voice</RecordButton>
      <Text>or</Text>
      <UploadButton>Upload Audio</UploadButton>
      <TargetVoiceSelector>
        <Label>Target Voice</Label>
        <VoiceDropdown />
      </TargetVoiceSelector>
    </VoiceChanger>
  )}
</AudioSourceSelector>
```

Integraciones necesarias:

- Conexión con 11 Labs API (para voces)
- Web Audio API para grabación
- Waveform visualizer para audio preview

D. ACTION/PROMPT INPUT

None

```
<ActionInput>
  <Label>Action/Movement</Label>
  <Textarea
    placeholder="Describe how you want your actor to move...
(e.g., 'talking naturally with relaxed hand gestures')"
```

```
    rows={3}
  />
  <QuickPrompts>
    <Chip>Talking to camera</Chip>
    <Chip>Natural hand gestures</Chip>
    <Chip>Looking excited</Chip>
    <Chip>Calm and relaxed</Chip>
  </QuickPrompts>
  <AIHelper>
    ✨ Generate prompt with AI
  </AIHelper>
</ActionInput>
```

Características:

- Sugerencias rápidas (chips clickeables)
- Auto-complete basado en prompts previos
- Tooltip con ejemplos al hacer hover

E. GENERATE BUTTON

None

```
<GenerateButton
  disabled={!canGenerate}
  loading={isGenerating}
  onClick={handleGenerate}
>
  {isGenerating ? (
    <>
      <Spinner />
      Generating... {progress}%
    </>
  ) : (
    <>
      <Icon>✨</Icon>
      Generate Video
    </>
  )}
```

```
</GenerateButton>
```

Estados:

- Default: Grande, color primario llamativo
- Hover: Elevación + brillo
- Disabled: Gris, cursor not-allowed
- Loading: Spinner + barra de progreso
- Success: Checkmark animado

Validaciones antes de habilitar:

- ✅ Imagen cargada
- ✅ Script no vacío (easy mode) o audio configurado (custom)
- ✅ Créditos disponibles

F. RIGHT PANEL - VIDEO PREVIEW

None

```
<VideoPreviewPanel>
  {isGenerating ? (
    <LoadingState>
      <Spinner size="large" />
      <ProgressBar value={progress} />
      <Status>
        {status} {/* e.g., "Processing audio...", "Generating
video...", "Finalizing..." */}
      </Status>
      <EstimatedTime>~{eta} remaining</EstimatedTime>
    </LoadingState>
  ) : generatedVideo ? (
    <VideoResult>
      <VideoPlayer
        src={generatedVideo.url}
        controls
        autoplay
        loop
      />
      <ActionButtons>
```

```
        <DownloadButton>
          <Icon>⬇️</Icon> Download
        </DownloadButton>
        <SaveButton>
          <Icon>💾</Icon> Save to Library
        </SaveButton>
        <RegenerateButton>
          <Icon>🔄</Icon> Regenerate
        </RegenerateButton>
      </ActionButtons>
      <VariationsSection>
        <Label>Generate Variations (4 at once)</Label>
        <GenerateVariationsButton>
          Generate 4 variations
        </GenerateVariationsButton>
      </VariationsSection>
    </VideoResult>
  ) : (
    <EmptyState>
      <Icon>🎬</Icon>
      <Text>Your video will appear here</Text>
    </EmptyState>
  )}
</VideoPreviewPanel>
```

Video Player specs:

- Custom controls (play, pause, scrub, volume, fullscreen)
- Loop toggle
- Download en múltiples formatos (MP4, WebM)
- Thumbnail preview al scrub

🖼️ 2. MAGIC EDIT (NANO BANANA) - EDITOR DE IMÁGENES

Interface Layout:

None

```
┌──────────────────────────────────────────────────────────┐
│  [<- Back]  Magic Edit              [Credits: 85] │
├─────────────┬────────────────────────────────────────┤
│             │                                        │
│   TOOLBAR   │      CANVAS AREA                       │
│             │   [Image being edited]                 │
│ [Edit Mode] │                                        │
│ [Product]   │                                        │
│ [Background]│                                        │
│ [Outfit]    │                                        │
│ [Remove]    │                                        │
│             │                                        │
│             ├────────────────────────────────────────┤
│             │   PROMPT PANEL                         │
│             │   [Textarea: "describe changes"]       │
│             │   [Reference Image Upload]             │
│             │   [Generate 4 variations]              │
└─────────────┴────────────────────────────────────────┘
```

A. CANVAS AREA

None

```
<ImageCanvas>
  <InteractiveImage
    src={sourceImage}
    onRegionSelect={handleRegionSelect}
    selectedRegion={selectedRegion}
  />
  {selectedRegion && (
    <SelectionOverlay>
      <BoundingBox region={selectedRegion} />
      <ResizeHandles />
    </SelectionOverlay>
  )}
</ImageCanvas>
```

Features:

- Zoom in/out (pinch, scroll wheel)

- Pan (drag cuando zoom > 100%)
- Selection tool para áreas específicas
- Undo/Redo stack
- Comparison slider (before/after)

B. EDIT MODES (Tabs laterales)

1. General Edit Mode:

```
None
<EditModePanel>
  <PromptInput>
    <Label>Describe your edits</Label>
    <Textarea
      placeholder="e.g., 'remove the hat', 'change background
to beach', 'add sunglasses'"
    />
  </PromptInput>

  <ReferenceImage>
    <Label>Reference Image (optional)</Label>
    <Upload />
  </ReferenceImage>

  <StrengthSlider>
    <Label>Edit Strength</Label>
    <Slider min={0} max={100} value={75} />
  </StrengthSlider>

  <GenerateButton>Generate 4 variations</GenerateButton>
</EditModePanel>
```

2. Product Mode:

```
None
<ProductPanel>
  <ProductUpload>
    <Label>Upload Product Image</Label>
    <DropZone>
      <Text>White background recommended</Text>
```

```
    </DropZone>
  </ProductUpload>

  <InteractionSelect>
    <Label>How should actor interact?</Label>
    <Dropdown>
      <Option>Hold in hand</Option>
      <Option>Wear on body</Option>
      <Option>Use/Apply</Option>
      <Option>Point at</Option>
    </Dropdown>
  </InteractionSelect>

  <PromptInput
    placeholder="Additional instructions..."
  />

  <GenerateButton>Add Product</GenerateButton>
</ProductPanel>
```

3. Background Mode:

None

```
<BackgroundPanel>
  <Tabs>
    <Tab>Describe</Tab>
    <Tab>Upload</Tab>
    <Tab>Presets</Tab>
  </Tabs>

  {tab === 'describe' && (
    <DescribeInput
      placeholder="e.g., 'cozy coffee shop', 'modern office',
'outdoor park'"
    />
  )}

  {tab === 'upload' && (
    <BackgroundUpload />
```

```
  )}

  {tab === 'presets' && (
    <PresetGrid>
      <PresetCard bg="studio-white">Studio White</PresetCard>
      <PresetCard bg="outdoor-park">Outdoor Park</PresetCard>
      <PresetCard bg="coffee-shop">Coffee Shop</PresetCard>
      {/* etc */}
    </PresetGrid>
  )}
</BackgroundPanel>
```

4. Outfit Mode:

None

```
<OutfitPanel>
  <OutfitDescription>
    <Label>Describe new outfit</Label>
    <Textarea
      placeholder="e.g., 'casual white t-shirt and jeans',
'business suit', 'yoga outfit'"
    />
  </OutfitDescription>

  <StylePresets>
    <Chip>Casual</Chip>
    <Chip>Business</Chip>
    <Chip>Athletic</Chip>
    <Chip>Formal</Chip>
  </StylePresets>

  <ReferenceOutfit>
    <Label>Reference Image</Label>
    <Upload />
  </ReferenceOutfit>
</OutfitPanel>
```

5. Remove Mode:

```
None
<RemovePanel>
  <BrushTool>
    <Label>Brush Size</Label>
    <Slider min={5} max={100} />
  </BrushTool>

  <Instructions>
    Paint over what you want to remove
  </Instructions>

  <MaskCanvas
    onPaint={handlePaint}
    brushSize={brushSize}
  />

  <Actions>
    <ClearButton>Clear Mask</ClearButton>
    <RemoveButton>Remove Selected</RemoveButton>
  </Actions>
</RemovePanel>
```

C. VARIATIONS GRID (Results)

```
None
<VariationsGrid>
  {variations.map(variation => (
    <VariationCard key={variation.id}>
      <ImagePreview src={variation.url} />
      <HoverActions>
        <IconButton title="Use This">✓</IconButton>
        <IconButton title="Download">⬇</IconButton>
        <IconButton title="Edit Further">✏️</IconButton>
        <IconButton title="Save to Actors">💾</IconButton>
      </HoverActions>
    </VariationCard>
  ))}
</VariationsGrid>
```

Comportamiento:

- Grid responsivo (2x2 en desktop, 1 col en mobile)
- Lazy loading
- Hover effects para revelar acciones
- Click para expandir/comparar
- Botón "Save to Actor Library" prominente

🎬 3. AI AVATAR GENERATOR

Interface Simple:

```
None
<AvatarGenerator>
  <PromptSection>
    <Label>Describe your actor</Label>
    <Textarea
      placeholder="e.g., '26-year-old female wellness creator,
natural lighting, minimal makeup, yoga mat in background'"
      rows={4}
    />
    <Examples>
      <ExampleChip>Man taking selfie in park</ExampleChip>
      <ExampleChip>Woman in coffee shop</ExampleChip>
      <ExampleChip>Professional headshot</ExampleChip>
    </Examples>
  </PromptSection>

  <AdvancedOptions collapsed>
    <Accordion title="Advanced Options">
      <Select label="Ethnicity" />
      <Select label="Age Range" />
      <Select label="Gender" />
      <Input label="Specific outfit" />
      <Input label="Background details" />
    </Accordion>
  </AdvancedOptions>

  <GenerateButton>Generate Actor</GenerateButton>
```

```
  <ResultsGrid>
    {results.map(actor => (
      <ActorCard>
        <Image src={actor.url} />
        <Actions>
          <EditButton>Edit</EditButton>
          <SaveButton>Save to Library</SaveButton>
          <UseButton>Use in Video</UseButton>
        </Actions>
      </ActorCard>
    ))}
  </ResultsGrid>
</AvatarGenerator>
```

📚 4. ACTOR LIBRARY

None

```
<ActorLibrary>
  <Header>
    <Tabs>
      <Tab active>Speel Library</Tab>
      <Tab>My Actors</Tab>
    </Tabs>
    <Search placeholder="Search actors..." />
    <Filters>
      <FilterDropdown label="Gender" />
      <FilterDropdown label="Age" />
      <FilterDropdown label="Ethnicity" />
      <FilterDropdown label="Style" />
    </Filters>
  </Header>

  <ActorGrid>
    {actors.map(actor => (
      <ActorCard>
        <Thumbnail src={actor.thumbnail} />
        <Info>
```

```
          <Tags>
            <Tag>{actor.gender}</Tag>
            <Tag>{actor.age}</Tag>
            <Tag>{actor.style}</Tag>
          </Tags>
        </Info>
        <SelectButton>Select Actor</SelectButton>
      </ActorCard>
    ))}
  </ActorGrid>

  <Pagination />
</ActorLibrary>
```

Features:

- Infinite scroll o paginación
- Preview al hover
- Multi-select para comparación
- Favoritos/Starred
- Recently used section

🎞️ 5. SCENES AND B-ROLL (Sora 2)

None

```
<ScenesGenerator>
  <ModelSelector>
    <Label>Engine</Label>
    <Dropdown value="sora-2-pro">
      <Option>Sora 2 Pro</Option>
      <Option>Cling 2.6</Option>
      <Option>Other models...</Option>
    </Dropdown>
  </ModelSelector>

  <Settings>
    <Select label="Duration">
```

```
      <Option>12 seconds</Option>
    </Select>
    <Select label="Resolution">
      <Option>1080p</Option>
    </Select>
    <Select label="Aspect Ratio">
      <Option>9:16 (Portrait)</Option>
      <Option>16:9 (Landscape)</Option>
      <Option>1:1 (Square)</Option>
    </Select>
  </Settings>

  <PromptSection>
    <Label>Scene Description</Label>
    <Textarea
      placeholder="Describe the scene... e.g., 'FaceTime style
video, close-up product shots applying the product, never show
clear face'"
    />
    <StyleChips>
      <Chip>FaceTime style</Chip>
      <Chip>TikTok native</Chip>
      <Chip>Instagram Reels</Chip>
    </StyleChips>
  </PromptSection>

  <ReferenceImageSection>
    <Label>Reference Image (Product)</Label>
    <Upload
      accept="image/*"
      hint="White background recommended"
    />
  </ReferenceImageSection>

  <GenerateButton>Generate Scene</GenerateButton>
</ScenesGenerator>
```

📁 6. IMAGES LIBRARY

None

```
<ImagesLibrary>
  <Toolbar>
    <Search />
    <FilterBar>
      <Filter>All</Filter>
      <Filter>Generated</Filter>
      <Filter>Uploaded</Filter>
      <Filter>Edited</Filter>
    </FilterBar>
    <SortDropdown>
      <Option>Newest First</Option>
      <Option>Oldest First</Option>
      <Option>Name A-Z</Option>
    </SortDropdown>
    <ViewToggle>
      <Icon active>Grid</Icon>
      <Icon>List</Icon>
    </ViewToggle>
  </Toolbar>

  <ImageGrid>
    {images.map(img => (
      <ImageCard>
        <Thumbnail src={img.url} />
        <Checkbox />
        <HoverMenu>
          <MenuItem>Edit in Magic Edit</MenuItem>
          <MenuItem>Use in Video</MenuItem>
          <MenuItem>Save to Actors</MenuItem>
          <MenuItem>Download</MenuItem>
          <MenuItem>Delete</MenuItem>
        </HoverMenu>
        <MetaInfo>
          <Date>{img.created}</Date>
          <Resolution>{img.resolution}</Resolution>
        </MetaInfo>
      </ImageCard>
    ))}
  </ImageGrid>
```

```
  <BulkActions visible={selectedImages.length > 0}>
    <Button>Download Selected</Button>
    <Button>Delete Selected</Button>
    <Button>Create Video from Selected</Button>
  </BulkActions>
</ImagesLibrary>
```

📽️ 7. VIDEOS LIBRARY

None

```
<VideosLibrary>
  <Toolbar>
    <Search />
    <Filters>
      <Filter>All Videos</Filter>
      <Filter>Easy Mode</Filter>
      <Filter>Custom Mode</Filter>
      <Filter>Sora 2</Filter>
    </Filters>
    <SortBy />
  </Toolbar>

  <VideoGrid>
    {videos.map(video => (
      <VideoCard>
        <VideoThumbnail
          src={video.thumbnail}
          onHover={playPreview}
        />
        <Duration>{video.duration}s</Duration>
        <Title>{video.title || 'Untitled'}</Title>
        <Meta>
          <Date>{video.created}</Date>
          <Model>{video.model}</Model>
        </Meta>
        <Actions>
```

```
          <PlayButton />
          <DownloadButton />
          <EditButton />
          <DeleteButton />
        </Actions>
      </VideoCard>
    ))}
  </VideoGrid>
</VideosLibrary>
```

Features:

- Video preview on hover
- Bulk download
- Export to different formats
- Share link generation
- Duplicate/Remix feature

⚙️ 8. COMPONENTES TÉCNICOS CRÍTICOS

A. CREDIT SYSTEM (Siempre visible)

```
None
<CreditDisplay position="top-right">
  <Icon>⭐</Icon>
  <Count>{remainingCredits}</Count>
  <ProgressBar
    value={usedCredits}
    max={totalCredits}
  />
  <Tooltip>
    <CreditBreakdown>
      <Item>Videos: {videoCredits} remaining</Item>
      <Item>Images: {imageCredits} remaining</Item>
      <Item>Resets in: {resetDate}</Item>
    </CreditBreakdown>
    <UpgradeButton>Upgrade Plan</UpgradeButton>
  </Tooltip>
```

```
</CreditDisplay>
```

B. PROGRESS TRACKING

```
None
<ProgressTracker>
  <Steps>
    <Step active completed>
      <Icon>✓</Icon>
      <Label>Processing Audio</Label>
    </Step>
    <Step active>
      <Spinner />
      <Label>Generating Video</Label>
      <Progress>47%</Progress>
    </Step>
    <Step>
      <Icon>⏳</Icon>
      <Label>Finalizing</Label>
    </Step>
  </Steps>
  <EstimatedTime>~2 minutes remaining</EstimatedTime>
  <CancelButton>Cancel Generation</CancelButton>
</ProgressTracker>
```

Estados de generación:

1. "Analyzing input..." (5%)
2. "Processing audio..." (15%)
3. "Generating frames..." (40%)
4. "Rendering video..." (80%)
5. "Finalizing..." (95%)
6. "Complete!" (100%)

C. NOTIFICATION SYSTEM

```
None
<NotificationCenter>
  <Toast type="success">
    Video generated successfully!
    <ViewButton />
  </Toast>

  <Toast type="error">
    Generation failed. Credits refunded.
    <RetryButton />
  </Toast>

  <Toast type="info">
    Your video is ready to download
    <DownloadButton />
  </Toast>

  <Toast type="warning">
    Low credits remaining (5 videos left)
    <UpgradeButton />
  </Toast>
</NotificationCenter>
```

Posición: Top-right, stack vertical
Auto-dismiss: 5 segundos (excepto errores)
Actions: Botones inline para acciones rápidas

D. MODAL SYSTEM

Script Helper Modal:

```
None
<ScriptHelperModal>
  <Header>
    <Title>AI Script Generator</Title>
    <CloseButton />
  </Header>
  <Body>
    <QuestionForm>
```

```
      <Question>What is your product/service?</Question>
      <Input />

      <Question>Who is your target audience?</Question>
      <Input />

      <Question>What problem does it solve?</Question>
      <Textarea />

      {/* more questions */}
    </QuestionForm>
  </Body>
  <Footer>
    <GenerateButton>Generate Script</GenerateButton>
  </Footer>
</ScriptHelperModal>
```

Voice Preview Modal:

```
None
<VoicePreviewModal>
  <VoiceInfo>
    <Avatar />
    <Name>Tessa - American Female</Name>
    <Tags>
      <Tag>English US</Tag>
      <Tag>Friendly</Tag>
      <Tag>Young Adult</Tag>
    </Tags>
  </VoiceInfo>
  <AudioPlayer>
    <Waveform />
    <Controls />
  </AudioPlayer>
  <TestScript>
    <Label>Test with your script</Label>
    <Textarea />
    <GeneratePreviewButton />
  </TestScript>
```

```
  <SelectButton>Use This Voice</SelectButton>
</VoicePreviewModal>
```

🎨 9. DESIGN SYSTEM

Colores (según videos):

```css
:root {
  /* Primary */
  --primary: #6366F1; /* Indigo */
  --primary-hover: #4F46E5;
  --primary-light: #A5B4FC;

  /* Secondary */
  --secondary: #8B5CF6; /* Purple */

  /* Neutrals */
  --bg-main: #0F172A; /* Dark blue-gray */
  --bg-secondary: #1E293B;
  --bg-card: #334155;
  --text-primary: #F1F5F9;
  --text-secondary: #94A3B8;
  --border: #334155;

  /* Status */
  --success: #10B981;
  --error: #EF4444;
  --warning: #F59E0B;
  --info: #3B82F6;
}
```

Typography:

```css
/* Headers */
```

```
h1 { font-size: 2rem; font-weight: 700; }
h2 { font-size: 1.5rem; font-weight: 600; }
h3 { font-size: 1.25rem; font-weight: 600; }

/* Body */
body { font-size: 0.875rem; font-family: 'Inter', sans-serif;
}
.small { font-size: 0.75rem; }

/* Code/Mono */
code { font-family: 'Fira Code', monospace; }
```

Spacing:

```css
--spacing-xs: 0.25rem;  /* 4px */
--spacing-sm: 0.5rem;   /* 8px */
--spacing-md: 1rem;     /* 16px */
--spacing-lg: 1.5rem;   /* 24px */
--spacing-xl: 2rem;     /* 32px */
--spacing-2xl: 3rem;    /* 48px */
```

Border Radius:

```css
--radius-sm: 0.25rem;   /* 4px - tags, chips */
--radius-md: 0.5rem;    /* 8px - buttons, inputs */
--radius-lg: 0.75rem;   /* 12px - cards */
--radius-xl: 1rem;      /* 16px - modals */
```

Shadows:

```css
--shadow-sm: 0 1px 2px rgba(0,0,0,0.05);
--shadow-md: 0 4px 6px rgba(0,0,0,0.1);
--shadow-lg: 0 10px 15px rgba(0,0,0,0.2);
--shadow-xl: 0 20px 25px rgba(0,0,0,0.3);
```

🔄 10. ESTADOS Y FLUJOS

A. FLUJO DE GENERACIÓN DE VIDEO

```
None
1. User uploads image
   ↓
2. User enters script/audio
   ↓
3. User clicks "Generate"
   ↓
4. Validation:
   - Check credits
   - Validate inputs
   - Estimate cost
   ↓
5. Show confirmation modal
   "This will use X credits. Continue?"
   ↓
6. API call starts
   ↓
7. Progress updates (websocket/polling):
   - 0%: "Starting..."
   - 15%: "Processing audio..."
   - 40%: "Generating frames..."
   - 80%: "Rendering video..."
   - 95%: "Finalizing..."
   ↓
8. Success:
   - Show video player
   - Enable download
   - Show success toast
   - Update credits
   OR
   Error:
   - Show error message
   - Refund credits
   - Offer retry
```

B. FLUJO DE EDICIÓN DE IMAGEN

```
None
1. User selects image from library
   OR uploads new
   ↓
2. Opens in Magic Edit
   ↓
3. User selects edit mode:
   - General edit
   - Product
   - Background
   - Outfit
   - Remove
   ↓
4. User provides input:
   - Prompt
   - Reference image
   - Selection/mask
   ↓
5. Click "Generate"
   ↓
6. System generates 4 variations
   ↓
7. User reviews grid
   ↓
8. User selects favorite:
   - Save to library
   - Save to actors
   - Edit further
   - Use in video
```

📱 11. RESPONSIVE BEHAVIOR

Desktop (>1280px):

- Dual-column layout (inputs left, preview right)
- Full toolbars visible
- Grid: 4 columns para libraries

Tablet (768px - 1280px):

- Stacked layout (inputs top, preview bottom)
- Collapsible toolbars
- Grid: 2-3 columns

Mobile (<768px):

- Single column
- Bottom sheet para opciones
- Full-screen modals
- Grid: 1-2 columns
- Sticky generate button at bottom

🔌 12. INTEGRACIONES NECESARIAS

APIs y Servicios:

1. **Video Generation:**
    - Modelo interno de Speel (V3.1 Easy Mode)
    - Custom mode engine
    - Sora 2 Pro API
2. **Audio:**
    - 11 Labs API (text-to-speech, voice cloning)
    - Web Audio API (recording)
3. **Image Generation:**
    - Nano Banana (propio)
    - DALL-E o Midjourney API (avatar generator)
4. **Storage:**
    - AWS S3 / Cloudflare R2 (media storage)
    - CDN para delivery
5. **Real-time:**
    - WebSocket para progress updates
    - Server-Sent Events alternativa
6. **Payment:**
    - Stripe (subscriptions + credits)

⚡ 13. PERFORMANCE REQUIREMENTS

Loading States:

- Skeleton screens para content loading
- Progressive image loading (blur-up)
- Lazy load para grids (intersection observer)
- Virtual scrolling para listas largas (>100 items)

Optimizations:

- Video thumbnails pre-generated
- Image optimization (WebP, responsive sizes)
- Code splitting por ruta
- Service worker para assets caching

Real-time Updates:

- Polling interval: 2-3 seconds durante generation
- WebSocket preferido para <100ms latency
- Fallback a long-polling

🧩 14. KEY UI COMPONENTS LIBRARY

```
None
// Base components necesarios:

<Button
  variant="primary|secondary|ghost|danger"
  size="sm|md|lg"
  loading={boolean}
  disabled={boolean}
  icon={ReactNode}
/>

<Input
  type="text|email|password"
  label={string}
  error={string}
  helper={string}
/>
```

```
<Textarea
  rows={number}
  maxLength={number}
  showCounter={boolean}
/>

<Dropdown
  options={array}
  searchable={boolean}
  multi={boolean}
/>

<Slider
  min={number}
  max={number}
  step={number}
  value={number}
  onChange={function}
/>

<Toggle
  checked={boolean}
  onChange={function}
/>

<Tabs
  tabs={array}
  activeTab={string}
  onChange={function}
/>

<Card
  elevation="sm|md|lg"
  hoverable={boolean}
/>

<Modal
  open={boolean}
```

```
  onClose={function}
  size="sm|md|lg|xl|full"
/>

<Toast
  type="success|error|warning|info"
  duration={number}
  action={ReactNode}
/>

<ProgressBar
  value={number}
  max={number}
  animated={boolean}
/>

<Spinner
  size="sm|md|lg"
/>

<FileUpload
  accept={string}
  multiple={boolean}
  dragDrop={boolean}
  onUpload={function}
/>

<VideoPlayer
  src={string}
  controls={boolean}
  autoplay={boolean}
  loop={boolean}
/>

<Waveform
  audioSrc={string}
  onSeek={function}
/>
```

📋 15. USER FLOWS CRÍTICOS

First-time User Onboarding:

```
None
1. Sign up → 2. Welcome modal → 3. Quick tutorial (tooltips) →
4. First video generation (guided)
```

Tutorial tooltips:

- "Upload an image of your actor here"
- "Enter what you want them to say"
- "Click generate to create your first video!"

Credit Management:

```
None
<CreditWarningFlow>
  {credits < 10 && (
    <Banner type="warning">
      Running low on credits!
      <UpgradeButton />
    </Banner>
  )}

  {credits === 0 && (
    <BlockModal>
      <Title>Out of Credits</Title>
      <Text>You've used all your video credits this
month.</Text>
      <Options>
        <Button>Upgrade Plan</Button>
        <Button variant="secondary">Buy One-time
Credits</Button>
      </Options>
    </BlockModal>
  )}
</CreditWarningFlow>
```

🎯 16. FEATURES ÚNICAS A IMPLEMENTAR

A. Comparison Slider:

```
None
<ComparisonSlider>
  <ImageBefore src={original} />
  <ImageAfter src={edited} />
  <Slider
    onChange={setPosition}
    defaultValue={50}
  />
</ComparisonSlider>
```

Uso:

- Comparar before/after en Magic Edit
- Comparar diferentes variaciones

B. Batch Generation:

```
None
<BatchGenerator>
  <ScriptVariations>
    <Script>Hook variant 1</Script>
    <Script>Hook variant 2</Script>
    <Script>Hook variant 3</Script>
  </ScriptVariations>

  <ActorSelection mode="multiple">
    {/* Select 3-4 actors */}
  </ActorSelection>

  <GenerateAllButton>
    Generate {scripts.length × actors.length} videos
    <CostEstimate>{totalCredits} credits</CostEstimate>
  </GenerateAllButton>
</BatchGenerator>
```

C. Shot List Builder:

None

```
<ShotListBuilder>
  <ShotTimeline>
    <Shot duration={3}>
      <Thumbnail />
      <Script>Hook</Script>
    </Shot>
    <Shot duration={5}>
      <Thumbnail />
      <Script>Problem statement</Script>
    </Shot>
    <Shot duration={4}>
      <Thumbnail />
      <Script>Solution</Script>
    </Shot>
  </ShotTimeline>

  <AddShotButton />
  <ReorderHandle />
  <GenerateFullAdButton>
    Generate complete ad ({totalDuration}s)
  </GenerateFullAdButton>
</ShotListBuilder>
```

🚀 17. TECH STACK RECOMENDADO

JSON

```
{
  "frontend": {
    "framework": "Next.js 14+ (App Router)",
    "ui": "React 18+",
    "styling": "Tailwind CSS + Shadcn/ui",
    "state": "Zustand o Redux Toolkit",
    "forms": "React Hook Form + Zod",
    "video": "Video.js o Plyr",
    "drag-drop": "dnd-kit",
    "animations": "Framer Motion",
    "charts": "Recharts (si analytics)"
```

```
  },
  "backend": {
    "api": "Next.js API Routes o tRPC",
    "realtime": "Socket.io o Pusher",
    "queue": "Bull/BullMQ (job processing)",
    "storage": "AWS S3 / R2",
    "database": "PostgreSQL + Prisma"
  },
  "ai-integration": {
    "video-gen": "Custom API wrapper",
    "voice": "11 Labs SDK",
    "image-edit": "Replicate API o custom"
  }
}
```

✅ CHECKLIST PARA TU DEVELOPER

Phase 1: Core UI (Week 1-2)

- Setup Next.js + Tailwind + Shadcn
- Design system (colores, spacing, typography)
- Base components library
- Navigation structure
- Responsive layouts

Phase 2: UGC Builder (Week 3-4)

- Image upload component
- Easy Mode interface
- Custom Mode interface
- Voice selector con preview
- Script textarea con AI helper
- Generate button con estados
- Video preview player
- Download functionality

Phase 3: Magic Edit (Week 5-6)

- Canvas area con zoom/pan
- Selection tools
- Edit mode tabs

- Prompt inputs
- Product mode
- Background mode
- Variations grid
- Save to actors

Phase 4: Libraries (Week 7)

- Images library grid
- Videos library grid
- Search y filters
- Bulk actions
- Actor library

Phase 5: Integration (Week 8-9)

- API client setup
- WebSocket/SSE para progress
- 11 Labs integration
- Credit system
- Payment flow (Stripe)
- User auth

Phase 6: Polish (Week 10)

- Loading states everywhere
- Error handling
- Toast notifications
- Onboarding flow
- Mobile optimization
- Performance audit

📊 MÉTRICAS DE UX A TRACKEAR

None

```
// Analytics events importantes:

trackEvent('video_generation_started', {
  mode: 'easy' | 'custom',
  duration: number,
  hasReferenceImage: boolean
});
```

```
trackEvent('video_generation_completed', {
  generationTime: seconds,
  creditsUsed: number
});

trackEvent('video_downloaded', {
  format: 'mp4' | 'webm',
  source: 'direct' | 'library'
});

trackEvent('actor_saved_to_library', {
  source: 'generated' | 'edited'
});

trackEvent('credits_depleted');
trackEvent('upgrade_clicked');
```

🎁 BONUS: MICRO-INTERACTIONS

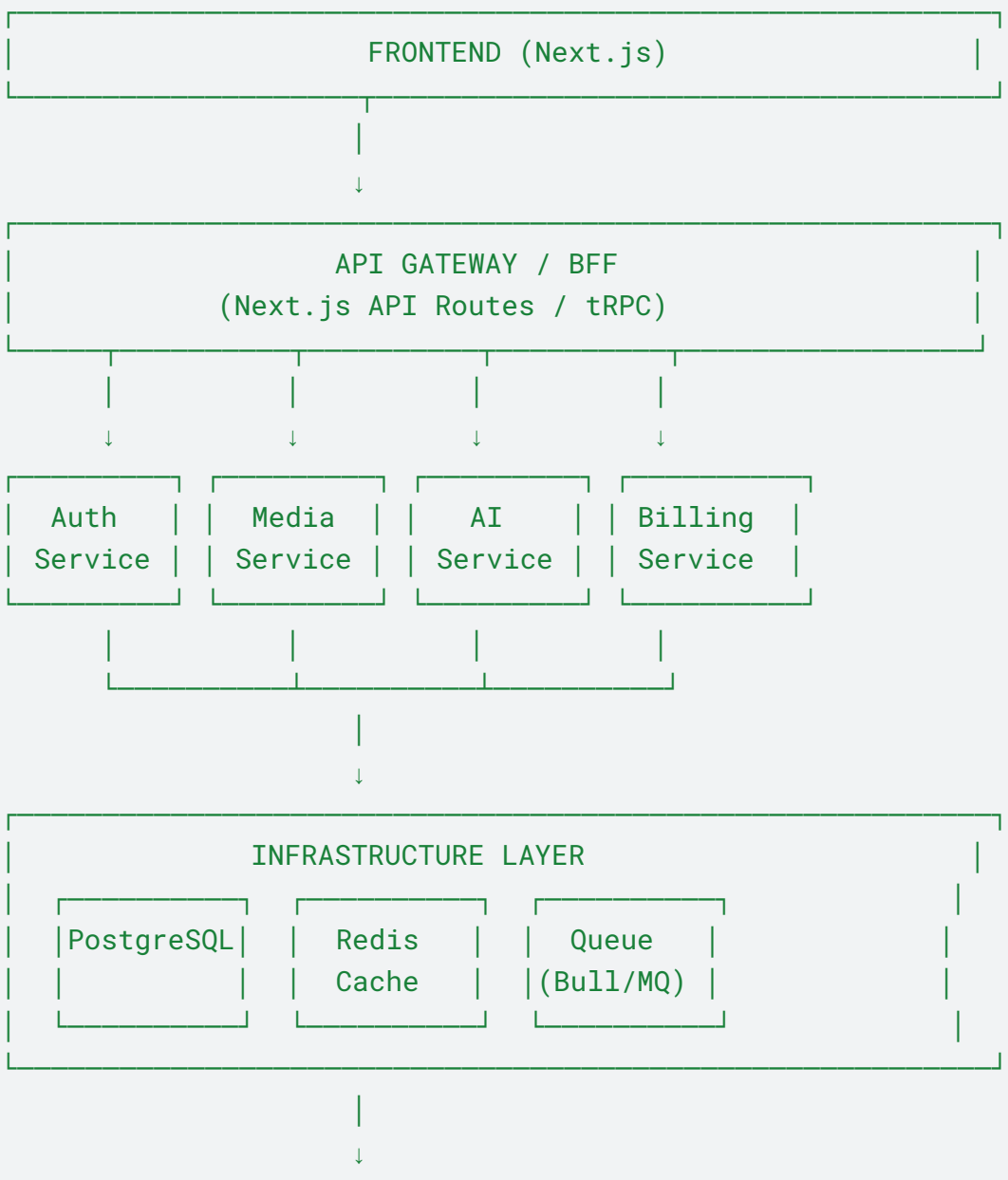
1. **Upload Success:** Checkmark animation + subtle bounce
2. **Generate Button:** Ripple effect + scale on click
3. **Video Ready:** Confetti animation (1 segundo)
4. **Credit Update:** Number counter animation
5. **Tab Switch:** Slide transition
6. **Card Hover:** Lift elevation + subtle glow
7. **Progress Bar:** Smooth fill animation + pulse
8. **Error State:** Shake animation

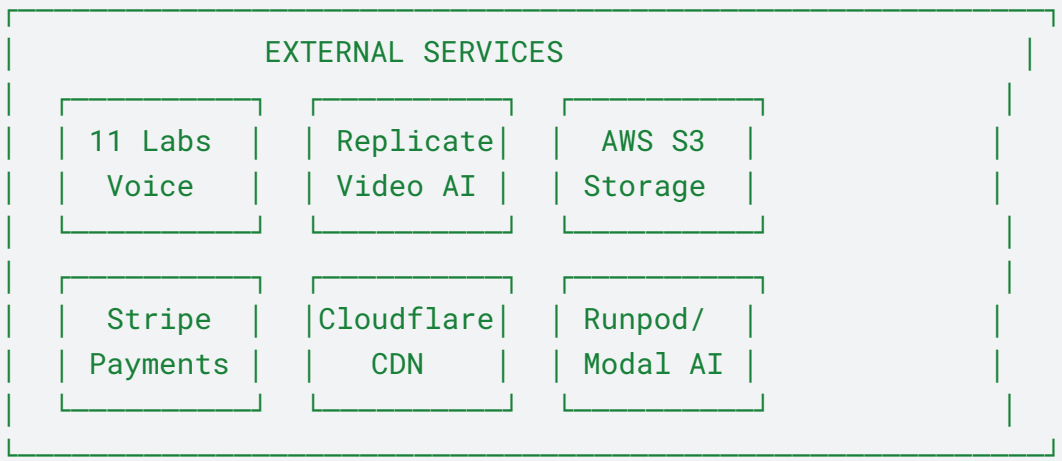
BACK-END

ESPECIFICACIONES TÉCNICAS COMPLETAS DEL BACKEND PARA SPEEL.CO

🏗️ ARQUITECTURA GENERAL DEL SISTEMA

None





🛠️ TECH STACK RECOMENDADO

None

```
Core Backend:
  - Runtime: Node.js 20+ (LTS)
  - Framework: Next.js 14+ (API Routes) o Express.js
  - Language: TypeScript (strict mode)
  - API Protocol: REST + tRPC (type-safe)

Database:
  - Primary: PostgreSQL 15+ (relational data)
  - Cache: Redis 7+ (sessions, rate limiting, job status)
  - Search: PostgreSQL Full-Text o Elasticsearch (optional)

Queue & Jobs:
  - Job Queue: BullMQ (Redis-based)
  - Scheduler: node-cron o Agenda
  - Worker Processes: Separate Node processes

Storage:
  - Object Storage: AWS S3 o Cloudflare R2
  - CDN: Cloudflare o AWS CloudFront
  - Temp Storage: Local disk + automatic cleanup

Real-time:
```

```
  - WebSockets: Socket.io o ws
  - Alternative: Server-Sent Events (SSE)
  - Pub/Sub: Redis Pub/Sub

Authentication:
  - Auth Provider: NextAuth.js o Clerk
  - JWT: jsonwebtoken
  - Session Store: Redis
  - OAuth: Google, GitHub (optional)

Payment:
  - Payment Gateway: Stripe
  - Webhooks: Stripe webhook handlers
  - Subscription Management: Stripe Subscriptions

AI/ML Integration:
  - Video Generation: Replicate API, RunPod, Modal
  - Image Generation: Replicate (SDXL, Flux)
  - Image Editing: Replicate (ControlNet, InstantID)
  - Voice: 11 Labs API
  - Lip Sync: Custom model o Wav2Lip

Monitoring:
  - APM: Sentry o Datadog
  - Logs: Winston + Logtail
  - Metrics: Prometheus + Grafana
  - Uptime: BetterStack

DevOps:
  - Hosting: Vercel (frontend) + Railway/Fly.io (backend)
  - CI/CD: GitHub Actions
  - Infrastructure: Docker + Docker Compose
  - Secrets: Doppler o AWS Secrets Manager
```

📊 DATABASE SCHEMA (PostgreSQL)

SQL

```sql
-- ============================================
-- USERS & AUTHENTICATION
-- ============================================

CREATE TABLE users (
  id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
  email VARCHAR(255) UNIQUE NOT NULL,
  password_hash VARCHAR(255), -- nullable if OAuth only
  name VARCHAR(255),
  avatar_url TEXT,
  email_verified BOOLEAN DEFAULT FALSE,

  -- Subscription
  subscription_tier VARCHAR(50) DEFAULT 'free', -- free,
starter, growth, pro
  subscription_status VARCHAR(50) DEFAULT 'active', -- active,
cancelled, expired
  subscription_id VARCHAR(255), -- Stripe subscription ID

  -- Credits
  video_credits_remaining INT DEFAULT 0,
  image_credits_remaining INT DEFAULT 0,
  credits_reset_date TIMESTAMP,

  -- Metadata
  created_at TIMESTAMP DEFAULT NOW(),
  updated_at TIMESTAMP DEFAULT NOW(),
  last_login_at TIMESTAMP,

  -- Settings
  preferences JSONB DEFAULT '{}'::jsonb
);

CREATE INDEX idx_users_email ON users(email);
CREATE INDEX idx_users_subscription_id ON
users(subscription_id);


-- ============================================
```

```sql
-- ACTORS & AVATARS
-- ============================================

CREATE TABLE actors (
  id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
  user_id UUID REFERENCES users(id) ON DELETE CASCADE,

  -- Actor Info
  name VARCHAR(255),
  description TEXT,
  thumbnail_url TEXT NOT NULL,
  full_image_url TEXT NOT NULL,

  -- Metadata
  gender VARCHAR(20), -- male, female, other
  age_range VARCHAR(20), -- 20s, 30s, 40s, etc.
  ethnicity VARCHAR(50),
  style VARCHAR(50), -- casual, business, athletic, etc.

  -- Source
  source VARCHAR(50) NOT NULL, -- generated, uploaded,
library, edited
  is_public BOOLEAN DEFAULT FALSE, -- for Speel Library
  is_favorite BOOLEAN DEFAULT FALSE,

  -- Generation params
  generation_prompt TEXT,
  generation_params JSONB,

  -- Usage
  usage_count INT DEFAULT 0,
  last_used_at TIMESTAMP,

  created_at TIMESTAMP DEFAULT NOW(),
  updated_at TIMESTAMP DEFAULT NOW()
);

CREATE INDEX idx_actors_user_id ON actors(user_id);
```

```
CREATE INDEX idx_actors_is_public ON actors(is_public) WHERE
is_public = TRUE;
CREATE INDEX idx_actors_source ON actors(source);


-- ============================================
-- IMAGES
-- ============================================

CREATE TABLE images (
  id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
  user_id UUID REFERENCES users(id) ON DELETE CASCADE,
  actor_id UUID REFERENCES actors(id) ON DELETE SET NULL,

  -- File info
  url TEXT NOT NULL,
  thumbnail_url TEXT,
  filename VARCHAR(255),
  file_size_bytes BIGINT,

  -- Image specs
  width INT,
  height INT,
  format VARCHAR(10), -- jpg, png, webp

  -- Source
  source VARCHAR(50) NOT NULL, -- uploaded, generated, edited
  parent_image_id UUID REFERENCES images(id), -- if edited
from another

  -- Generation/Edit data
  prompt TEXT,
  negative_prompt TEXT,
  model_used VARCHAR(100),
  generation_params JSONB,

  -- Metadata
  tags TEXT[], -- searchable tags
  is_favorite BOOLEAN DEFAULT FALSE,
```

```
  created_at TIMESTAMP DEFAULT NOW(),
  updated_at TIMESTAMP DEFAULT NOW()
);

CREATE INDEX idx_images_user_id ON images(user_id);
CREATE INDEX idx_images_actor_id ON images(actor_id);
CREATE INDEX idx_images_source ON images(source);
CREATE INDEX idx_images_created_at ON images(created_at DESC);


-- ============================================
-- VIDEOS
-- ============================================

CREATE TABLE videos (
  id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
  user_id UUID REFERENCES users(id) ON DELETE CASCADE,
  actor_id UUID REFERENCES actors(id) ON DELETE SET NULL,
  source_image_id UUID REFERENCES images(id) ON DELETE SET
NULL,

  -- File info
  url TEXT NOT NULL,
  thumbnail_url TEXT,
  filename VARCHAR(255),
  file_size_bytes BIGINT,

  -- Video specs
  duration_seconds DECIMAL(10, 2),
  width INT,
  height INT,
  fps INT DEFAULT 30,
  format VARCHAR(10), -- mp4, webm

  -- Generation info
  mode VARCHAR(50) NOT NULL, -- easy, custom, sora, cling
  model_used VARCHAR(100),
```

```
  -- Script/Audio
  script TEXT,
  voice_id VARCHAR(255), -- 11 Labs voice ID
  voice_name VARCHAR(255),
  audio_url TEXT,

  -- Action/Prompt
  action_prompt TEXT,
  generation_prompt TEXT,
  generation_params JSONB,

  -- Status
  status VARCHAR(50) DEFAULT 'pending', -- pending,
processing, completed, failed
  progress INT DEFAULT 0, -- 0-100
  error_message TEXT,

  -- Processing times
  started_at TIMESTAMP,
  completed_at TIMESTAMP,
  processing_duration_seconds INT,

  -- Metadata
  title VARCHAR(255),
  tags TEXT[],
  is_favorite BOOLEAN DEFAULT FALSE,

  created_at TIMESTAMP DEFAULT NOW(),
  updated_at TIMESTAMP DEFAULT NOW()
);

CREATE INDEX idx_videos_user_id ON videos(user_id);
CREATE INDEX idx_videos_actor_id ON videos(actor_id);
CREATE INDEX idx_videos_status ON videos(status);
CREATE INDEX idx_videos_mode ON videos(mode);
CREATE INDEX idx_videos_created_at ON videos(created_at DESC);


-- ============================================
```

```sql
-- VOICE CLONES (11 Labs)
-- ============================================

CREATE TABLE voice_clones (
  id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
  user_id UUID REFERENCES users(id) ON DELETE CASCADE,

  -- 11 Labs info
  voice_id VARCHAR(255) UNIQUE NOT NULL, -- 11 Labs voice ID
  voice_name VARCHAR(255) NOT NULL,

  -- Source
  source_audio_url TEXT,
  source_video_id UUID REFERENCES videos(id),

  -- Metadata
  description TEXT,
  accent VARCHAR(50),
  gender VARCHAR(20),
  age VARCHAR(20),

  -- Usage
  usage_count INT DEFAULT 0,
  last_used_at TIMESTAMP,

  created_at TIMESTAMP DEFAULT NOW()
);

CREATE INDEX idx_voice_clones_user_id ON
voice_clones(user_id);
CREATE INDEX idx_voice_clones_voice_id ON
voice_clones(voice_id);


-- ============================================
-- JOBS & PROCESSING
-- ============================================

CREATE TABLE processing_jobs (
```

```
  id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
  user_id UUID REFERENCES users(id) ON DELETE CASCADE,

  -- Job info
  job_type VARCHAR(50) NOT NULL, -- video_generation,
image_generation, image_edit, voice_clone
  job_id VARCHAR(255) UNIQUE NOT NULL, -- Bull queue job ID

  -- Related entities
  video_id UUID REFERENCES videos(id),
  image_id UUID REFERENCES images(id),

  -- Status
  status VARCHAR(50) DEFAULT 'pending', -- pending,
processing, completed, failed, cancelled
  progress INT DEFAULT 0,
  current_step VARCHAR(100),
  error_message TEXT,

  -- Timing
  started_at TIMESTAMP,
  completed_at TIMESTAMP,
  estimated_completion_at TIMESTAMP,

  -- Input/Output
  input_params JSONB,
  output_data JSONB,

  -- Costs
  credits_used INT DEFAULT 0,
  external_api_cost DECIMAL(10, 4), -- tracking actual costs

  created_at TIMESTAMP DEFAULT NOW(),
  updated_at TIMESTAMP DEFAULT NOW()
);

CREATE INDEX idx_jobs_user_id ON processing_jobs(user_id);
CREATE INDEX idx_jobs_job_id ON processing_jobs(job_id);
CREATE INDEX idx_jobs_status ON processing_jobs(status);
```

```sql
CREATE INDEX idx_jobs_video_id ON processing_jobs(video_id);


-- ============================================
-- CREDITS & BILLING
-- ============================================

CREATE TABLE credit_transactions (
  id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
  user_id UUID REFERENCES users(id) ON DELETE CASCADE,

  -- Transaction info
  type VARCHAR(50) NOT NULL, -- purchase,
subscription_renewal, usage, refund, bonus
  amount INT NOT NULL, -- positive for credits added, negative
for used
  credit_type VARCHAR(20) NOT NULL, -- video, image, voice

  -- Related entities
  video_id UUID REFERENCES videos(id),
  image_id UUID REFERENCES images(id),
  job_id UUID REFERENCES processing_jobs(id),

  -- Description
  description TEXT,

  -- Balances (snapshot)
  balance_before INT,
  balance_after INT,

  created_at TIMESTAMP DEFAULT NOW()
);

CREATE INDEX idx_credit_transactions_user_id ON
credit_transactions(user_id);
CREATE INDEX idx_credit_transactions_type ON
credit_transactions(type);
CREATE INDEX idx_credit_transactions_created_at ON
credit_transactions(created_at DESC);
```

```sql
CREATE TABLE subscription_plans (
  id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),

  -- Plan info
  name VARCHAR(100) NOT NULL, -- Starter, Growth, Pro
  tier VARCHAR(50) UNIQUE NOT NULL, -- starter, growth, pro

  -- Pricing
  price_monthly_usd DECIMAL(10, 2),
  price_yearly_usd DECIMAL(10, 2),

  -- Credits per month
  video_credits INT,
  image_credits INT,

  -- Limits
  max_video_duration_seconds INT,
  max_concurrent_generations INT,

  -- Features
  features JSONB, -- array of feature strings

  -- Stripe
  stripe_price_id_monthly VARCHAR(255),
  stripe_price_id_yearly VARCHAR(255),

  is_active BOOLEAN DEFAULT TRUE,

  created_at TIMESTAMP DEFAULT NOW(),
  updated_at TIMESTAMP DEFAULT NOW()
);


CREATE TABLE payment_transactions (
  id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
  user_id UUID REFERENCES users(id) ON DELETE CASCADE,
```

```sql
  -- Stripe info
  stripe_payment_intent_id VARCHAR(255) UNIQUE,
  stripe_invoice_id VARCHAR(255),

  -- Transaction details
  amount_usd DECIMAL(10, 2) NOT NULL,
  currency VARCHAR(3) DEFAULT 'USD',
  status VARCHAR(50) NOT NULL, -- succeeded, pending, failed,
refunded

  -- Type
  transaction_type VARCHAR(50), -- subscription,
one_time_credits, upgrade

  -- Credits granted
  video_credits_granted INT DEFAULT 0,
  image_credits_granted INT DEFAULT 0,

  -- Metadata
  description TEXT,
  metadata JSONB,

  created_at TIMESTAMP DEFAULT NOW()
);

CREATE INDEX idx_payment_transactions_user_id ON
payment_transactions(user_id);
CREATE INDEX idx_payment_transactions_stripe_payment_intent ON
payment_transactions(stripe_payment_intent_id);


-- ============================================
-- ANALYTICS & USAGE
-- ============================================

CREATE TABLE usage_analytics (
  id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
  user_id UUID REFERENCES users(id) ON DELETE CASCADE,
```

```
  -- Event
  event_type VARCHAR(100) NOT NULL, -- video_generated,
image_edited, actor_created, etc.
  event_data JSONB,

  -- Context
  session_id VARCHAR(255),
  ip_address INET,
  user_agent TEXT,

  -- Performance
  duration_ms INT,

  created_at TIMESTAMP DEFAULT NOW()
);

CREATE INDEX idx_usage_analytics_user_id ON
usage_analytics(user_id);
CREATE INDEX idx_usage_analytics_event_type ON
usage_analytics(event_type);
CREATE INDEX idx_usage_analytics_created_at ON
usage_analytics(created_at DESC);

-- Partitioning by month for analytics (optional but
recommended)
-- CREATE TABLE usage_analytics_2024_01 PARTITION OF
usage_analytics
-- FOR VALUES FROM ('2024-01-01') TO ('2024-02-01');


-- ============================================
-- API KEYS & INTEGRATIONS
-- ============================================

CREATE TABLE api_keys (
  id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
  user_id UUID REFERENCES users(id) ON DELETE CASCADE,

  -- Key info
```

```
  key_hash VARCHAR(255) UNIQUE NOT NULL, -- hashed API key
  key_prefix VARCHAR(20) NOT NULL, -- first chars for
identification
  name VARCHAR(255),

  -- Permissions
  scopes TEXT[], -- array of allowed operations

  -- Rate limiting
  rate_limit_per_minute INT DEFAULT 60,

  -- Status
  is_active BOOLEAN DEFAULT TRUE,
  last_used_at TIMESTAMP,

  -- Expiration
  expires_at TIMESTAMP,

  created_at TIMESTAMP DEFAULT NOW()
);

CREATE INDEX idx_api_keys_user_id ON api_keys(user_id);
CREATE INDEX idx_api_keys_key_hash ON api_keys(key_hash);
```

🔌 API ENDPOINTS & ARCHITECTURE

REST API Structure:

TypeScript

```
// ============================================
// AUTH ENDPOINTS
// ============================================

POST   /api/auth/register
POST   /api/auth/login
POST   /api/auth/logout
POST   /api/auth/refresh-token
```

```
POST   /api/auth/forgot-password
POST   /api/auth/reset-password
GET    /api/auth/me
PATCH  /api/auth/me

// OAuth
GET    /api/auth/google
GET    /api/auth/google/callback
GET    /api/auth/github
GET    /api/auth/github/callback


// ============================================
// USER ENDPOINTS
// ============================================

GET    /api/users/:userId
PATCH  /api/users/:userId
DELETE /api/users/:userId

// Credits & Subscription
GET    /api/users/:userId/credits
GET    /api/users/:userId/subscription
POST   /api/users/:userId/subscription/upgrade
POST   /api/users/:userId/subscription/cancel

// Usage
GET    /api/users/:userId/usage-stats
GET    /api/users/:userId/analytics


// ============================================
// ACTOR ENDPOINTS
// ============================================

GET    /api/actors                    // List user's actors
POST   /api/actors                    // Create/upload actor
GET    /api/actors/:actorId
PATCH  /api/actors/:actorId
```

```
DELETE /api/actors/:actorId

// Actor Library (public)
GET    /api/actors/library            // Speel's actor library
GET    /api/actors/library/:actorId

// Favorites
POST   /api/actors/:actorId/favorite
DELETE /api/actors/:actorId/favorite


// ============================================
// IMAGE ENDPOINTS
// ============================================

GET    /api/images                    // List with pagination
POST   /api/images/upload             // Upload image
POST   /api/images/generate           // Generate with AI
GET    /api/images/:imageId
PATCH  /api/images/:imageId           // Update metadata
DELETE /api/images/:imageId

// Image Editing (Nano Banana)
POST   /api/images/:imageId/edit      // General edit
POST   /api/images/:imageId/add-product
POST   /api/images/:imageId/change-background
POST   /api/images/:imageId/change-outfit
POST   /api/images/:imageId/remove

// Batch operations
POST   /api/images/batch-download
DELETE /api/images/batch-delete


// ============================================
// VIDEO ENDPOINTS
// ============================================

GET    /api/videos                    // List with filters
```

```
POST   /api/videos/generate           // Generate video
GET    /api/videos/:videoId
PATCH  /api/videos/:videoId           // Update metadata
DELETE /api/videos/:videoId

// Video modes
POST   /api/videos/generate/easy-mode
POST   /api/videos/generate/custom-mode
POST   /api/videos/generate/sora
POST   /api/videos/generate/cling

// Video operations
POST   /api/videos/:videoId/regenerate
GET    /api/videos/:videoId/download
POST   /api/videos/:videoId/variations  // Generate 4
variations

// Batch
POST   /api/videos/batch-generate      // Bulk generation


// ============================================
// VOICE ENDPOINTS
// ============================================

GET    /api/voices                     // List 11 Labs voices
GET    /api/voices/:voiceId/preview
POST   /api/voices/clone               // Clone voice
GET    /api/voices/my-clones
DELETE /api/voices/clones/:cloneId

// Text to Speech
POST   /api/voices/text-to-speech
POST   /api/voices/speech-to-speech


// ============================================
// JOB/PROCESSING ENDPOINTS
// ============================================
```

```
GET    /api/jobs/:jobId                 // Job status
GET    /api/jobs/:jobId/progress        // Real-time progress
POST   /api/jobs/:jobId/cancel
GET    /api/jobs                         // List user's jobs

// WebSocket alternative
WS     /api/jobs/:jobId/subscribe       // WebSocket for
real-time


// ============================================
// BILLING ENDPOINTS
// ============================================

GET    /api/billing/plans               // Available plans
POST   /api/billing/checkout            // Create Stripe
checkout
POST   /api/billing/portal              // Stripe customer
portal
GET    /api/billing/invoices
GET    /api/billing/transactions

// Webhooks
POST   /api/webhooks/stripe             // Stripe webhook
handler


// ============================================
// CREDITS ENDPOINTS
// ============================================

GET    /api/credits/balance
GET    /api/credits/transactions
POST   /api/credits/purchase            // One-time credits


// ============================================
// ANALYTICS ENDPOINTS
```

```
// ============================================

GET    /api/analytics/usage
GET    /api/analytics/costs
GET    /api/analytics/popular-actors
```

⚙️ CORE SERVICES ARCHITECTURE

1. AUTH SERVICE

TypeScript

```typescript
// services/auth.service.ts

import bcrypt from 'bcryptjs';
import jwt from 'jsonwebtoken';
import { User } from '@/types';

class AuthService {
  async register(email: string, password: string, name:
string): Promise<User> {
    // 1. Validate input
    // 2. Check if user exists
    // 3. Hash password
    const passwordHash = await bcrypt.hash(password, 12);

    // 4. Create user
    const user = await db.users.create({
      email,
      password_hash: passwordHash,
      name,
      subscription_tier: 'free',
      video_credits_remaining: 3, // Free trial
      image_credits_remaining: 10,
      credits_reset_date: addMonths(new Date(), 1)
    });

    // 5. Send verification email
```

```
    await emailService.sendVerification(user.email);

    return user;
  }

  async login(email: string, password: string): Promise<{
user: User; token: string }> {
    // 1. Find user
    const user = await db.users.findByEmail(email);
    if (!user) throw new Error('Invalid credentials');

    // 2. Verify password
    const isValid = await bcrypt.compare(password,
user.password_hash);
    if (!isValid) throw new Error('Invalid credentials');

    // 3. Update last login
    await db.users.update(user.id, { last_login_at: new Date()
});

    // 4. Generate JWT
    const token = jwt.sign(
      { userId: user.id, email: user.email },
      process.env.JWT_SECRET!,
      { expiresIn: '7d' }
    );

    return { user, token };
  }

  async verifyToken(token: string): Promise<User> {
    const decoded = jwt.verify(token,
process.env.JWT_SECRET!);
    const user = await db.users.findById(decoded.userId);
    return user;
  }
}

export const authService = new AuthService();
```

---

2. MEDIA SERVICE (Upload & Storage)

TypeScript

```
// services/media.service.ts

import { S3Client, PutObjectCommand, DeleteObjectCommand }
from '@aws-sdk/client-s3';
import sharp from 'sharp';
import { v4 as uuidv4 } from 'uuid';

class MediaService {
  private s3: S3Client;
  private bucket: string;
  private cdnUrl: string;

  constructor() {
    this.s3 = new S3Client({
      region: process.env.AWS_REGION,
      credentials: {
        accessKeyId: process.env.AWS_ACCESS_KEY_ID!,
        secretAccessKey: process.env.AWS_SECRET_ACCESS_KEY!
      }
    });
    this.bucket = process.env.S3_BUCKET!;
    this.cdnUrl = process.env.CDN_URL!;
  }

  async uploadImage(
    userId: string,
    file: Buffer,
    metadata: { filename: string; mimeType: string }
  ): Promise<{ url: string; thumbnailUrl: string; width:
number; height: number }> {

    // 1. Generate unique ID
    const imageId = uuidv4();
    const ext = metadata.filename.split('.').pop();

    // 2. Process image
```

```
    const processed = await sharp(file)
      .resize(2048, 2048, { fit: 'inside', withoutEnlargement:
true })
      .webp({ quality: 90 })
      .toBuffer();

    const imageMetadata = await sharp(processed).metadata();

    // 3. Generate thumbnail
    const thumbnail = await sharp(file)
      .resize(400, 400, { fit: 'cover' })
      .webp({ quality: 80 })
      .toBuffer();

    // 4. Upload to S3
    const imagePath =
`users/${userId}/images/${imageId}.webp`;
    const thumbnailPath =
`users/${userId}/images/thumbnails/${imageId}.webp`;

    await this.s3.send(new PutObjectCommand({
      Bucket: this.bucket,
      Key: imagePath,
      Body: processed,
      ContentType: 'image/webp',
      CacheControl: 'public, max-age=31536000',
      Metadata: {
        userId,
        originalFilename: metadata.filename
      }
    }));

    await this.s3.send(new PutObjectCommand({
      Bucket: this.bucket,
      Key: thumbnailPath,
      Body: thumbnail,
      ContentType: 'image/webp',
      CacheControl: 'public, max-age=31536000'
    }));
```

```
    return {
      url: `${this.cdnUrl}/${imagePath}`,
      thumbnailUrl: `${this.cdnUrl}/${thumbnailPath}`,
      width: imageMetadata.width!,
      height: imageMetadata.height!
    };
  }

  async uploadVideo(
    userId: string,
    videoBuffer: Buffer,
    metadata: { filename: string; duration: number }
  ): Promise<{ url: string; thumbnailUrl: string }> {

    const videoId = uuidv4();
    const videoPath = `users/${userId}/videos/${videoId}.mp4`;

    // 1. Upload video
    await this.s3.send(new PutObjectCommand({
      Bucket: this.bucket,
      Key: videoPath,
      Body: videoBuffer,
      ContentType: 'video/mp4',
      CacheControl: 'public, max-age=31536000'
    }));

    // 2. Generate thumbnail (use ffmpeg or external service)
    const thumbnail = await
this.generateVideoThumbnail(videoBuffer);
    const thumbnailPath =
`users/${userId}/videos/thumbnails/${videoId}.webp`;

    await this.s3.send(new PutObjectCommand({
      Bucket: this.bucket,
      Key: thumbnailPath,
      Body: thumbnail,
      ContentType: 'image/webp'
    }));
```

```
    return {
      url: `${this.cdnUrl}/${videoPath}`,
      thumbnailUrl: `${this.cdnUrl}/${thumbnailPath}`
    };
  }

  async deleteFile(path: string): Promise<void> {
    const key = path.replace(this.cdnUrl + '/', '');
    await this.s3.send(new DeleteObjectCommand({
      Bucket: this.bucket,
      Key: key
    }));
  }

  private async generateVideoThumbnail(videoBuffer: Buffer):
Promise<Buffer> {
    // Use ffmpeg or external service like Cloudinary
    // For now, placeholder:
    return Buffer.from('');
  }
}

export const mediaService = new MediaService();
```

---

3. AI SERVICE (Video & Image Generation)

TypeScript

```
// services/ai.service.ts

import Replicate from 'replicate';
import axios from 'axios';

class AIService {
  private replicate: Replicate;

  constructor() {
```

```
    this.replicate = new Replicate({
      auth: process.env.REPLICATE_API_TOKEN!
    });
  }

  // ==========================================
  // VIDEO GENERATION
  // ==========================================

  async generateVideoEasyMode(params: {
    imageUrl: string;
    script: string;
    actionPrompt?: string;
  }): Promise<{ jobId: string }> {

    // Using your own video model or Replicate's
    // Example with a hypothetical lip-sync model

    const output = await this.replicate.predictions.create({
      version: 'YOUR_VIDEO_MODEL_VERSION',
      input: {
        image: params.imageUrl,
        audio_prompt: params.script,
        motion_prompt: params.actionPrompt || 'natural talking
with hand gestures',
        duration: 8, // Easy mode = 8 seconds
        fps: 30
      },
      webhook:
`${process.env.APP_URL}/api/webhooks/replicate`,
      webhook_events_filter: ['completed']
    });

    return { jobId: output.id };
  }

  async generateVideoCustomMode(params: {
    imageUrl: string;
    audioUrl: string;
```

```
    actionPrompt?: string;
    duration?: number;
  }): Promise<{ jobId: string }> {

    const output = await this.replicate.predictions.create({
      version: 'YOUR_CUSTOM_VIDEO_MODEL_VERSION',
      input: {
        image: params.imageUrl,
        audio: params.audioUrl,
        motion_prompt: params.actionPrompt || 'natural
gestures',
        duration: params.duration || 60,
        fps: 30
      },
      webhook: `${process.env.APP_URL}/api/webhooks/replicate`
    });

    return { jobId: output.id };
  }

  async generateWithSora(params: {
    prompt: string;
    referenceImageUrl?: string;
    duration: number;
  }): Promise<{ jobId: string }> {

    // If you have access to Sora API
    // Otherwise use alternatives like Runway, Pika, etc.

    const response = await
axios.post('YOUR_SORA_API_ENDPOINT', {
      prompt: params.prompt,
      reference_image: params.referenceImageUrl,
      duration: params.duration,
      resolution: '1080p',
      aspect_ratio: '9:16'
    }, {
      headers: {
        'Authorization': `Bearer ${process.env.SORA_API_KEY}`
```

```
      }
    });

    return { jobId: response.data.job_id };
  }

  // ==========================================
  // IMAGE GENERATION
  // ==========================================

  async generateActor(params: {
    prompt: string;
    negativePrompt?: string;
    style?: string;
  }): Promise<string[]> {

    // Using SDXL or Flux for realistic portraits
    const output = await this.replicate.run(
      'stability-ai/sdxl:latest',
      {
        input: {
          prompt: params.prompt,
          negative_prompt: params.negativePrompt || 'cartoon,
anime, painting, illustration',
          num_outputs: 4,
          guidance_scale: 7.5,
          num_inference_steps: 50,
          width: 1024,
          height: 1024
        }
      }
    );

    return output as string[];
  }

  // ==========================================
  // IMAGE EDITING (Nano Banana equivalent)
  // ==========================================
```

```
  async editImage(params: {
    imageUrl: string;
    prompt: string;
    editType: 'general' | 'product' | 'background' | 'outfit'
| 'remove';
    referenceImageUrl?: string;
    maskUrl?: string;
  }): Promise<string[]> {

    // Using ControlNet, InstantID, or IP-Adapter

    if (params.editType === 'product') {
      return this.addProductToImage(params.imageUrl,
params.referenceImageUrl!, params.prompt);
    }

    if (params.editType === 'background') {
      return this.changeBackground(params.imageUrl,
params.prompt);
    }

    if (params.editType === 'outfit') {
      return this.changeOutfit(params.imageUrl,
params.prompt);
    }

    if (params.editType === 'remove') {
      return this.removeFromImage(params.imageUrl,
params.maskUrl!);
    }

    // General edit
    const output = await this.replicate.run(
      'stability-ai/stable-diffusion-inpainting',
      {
        input: {
          image: params.imageUrl,
          prompt: params.prompt,
```

```
        num_outputs: 4
      }
    }
  );

  return output as string[];
}

private async addProductToImage(
  actorImageUrl: string,
  productImageUrl: string,
  interactionPrompt: string
): Promise<string[]> {

  // Using IP-Adapter or ControlNet
  const output = await this.replicate.run(
    'YOUR_PRODUCT_INTEGRATION_MODEL',
    {
      input: {
        actor_image: actorImageUrl,
        product_image: productImageUrl,
        prompt: interactionPrompt,
        controlnet_conditioning_scale: 0.8,
        num_outputs: 4
      }
    }
  );

  return output as string[];
}

private async changeBackground(
  imageUrl: string,
  backgroundPrompt: string
): Promise<string[]> {

  // Using background removal + generation
  const output = await this.replicate.run(
    'cjwbw/rembg:latest', // Background removal
```

```
      { input: { image: imageUrl } }
    );

    const removedBgUrl = output[0];

    // Then composite with new background
    const final = await this.replicate.run(
      'YOUR_BACKGROUND_COMPOSITE_MODEL',
      {
        input: {
          foreground: removedBgUrl,
          background_prompt: backgroundPrompt,
          num_outputs: 4
        }
      }
    );

    return final as string[];
  }

  private async changeOutfit(
    imageUrl: string,
    outfitPrompt: string
  ): Promise<string[]> {

    // Using virtual try-on models
    const output = await this.replicate.run(
      'YOUR_OUTFIT_CHANGE_MODEL',
      {
        input: {
          image: imageUrl,
          outfit_description: outfitPrompt,
          preserve_face: true,
          num_outputs: 4
        }
      }
    );

    return output as string[];
```

```
  }

  private async removeFromImage(
    imageUrl: string,
    maskUrl: string
  ): Promise<string[]> {

    // Using inpainting
    const output = await this.replicate.run(
      'stability-ai/stable-diffusion-inpainting',
      {
        input: {
          image: imageUrl,
          mask: maskUrl,
          prompt: 'remove object, fill with background',
          num_outputs: 4
        }
      }
    );

    return output as string[];
  }

  // ========================================
  // STATUS CHECKING
  // ========================================

  async checkJobStatus(jobId: string): Promise<{
    status: 'pending' | 'processing' | 'completed' | 'failed';
    progress?: number;
    output?: any;
    error?: string;
  }> {

    const prediction = await
this.replicate.predictions.get(jobId);

    return {
      status: this.mapStatus(prediction.status),
```

```
      progress: this.calculateProgress(prediction),
      output: prediction.output,
      error: prediction.error?.toString()
    };
  }

  private mapStatus(status: string): 'pending' | 'processing'
| 'completed' | 'failed' {
    const mapping: Record<string, any> = {
      'starting': 'pending',
      'processing': 'processing',
      'succeeded': 'completed',
      'failed': 'failed',
      'canceled': 'failed'
    };
    return mapping[status] || 'pending';
  }

  private calculateProgress(prediction: any): number {
    // Estimate progress based on logs or metrics
    if (prediction.status === 'succeeded') return 100;
    if (prediction.status === 'failed') return 0;
    if (prediction.metrics?.predict_time) {
      // Rough estimation
      return Math.min(95, (prediction.metrics.predict_time /
120) * 100);
    }
    return 50; // Default
  }
}

export const aiService = new AIService();
```

---

4. VOICE SERVICE (11 Labs Integration)

TypeScript

```
// services/voice.service.ts
```

```
import axios from 'axios';
import FormData from 'form-data';

class VoiceService {
  private apiKey: string;
  private baseUrl = 'https://api.elevenlabs.io/v1';

  constructor() {
    this.apiKey = process.env.ELEVEN_LABS_API_KEY!;
  }

  async getVoices(): Promise<any[]> {
    const response = await axios.get(`${this.baseUrl}/voices`,
{
      headers: { 'xi-api-key': this.apiKey }
    });
    return response.data.voices;
  }

  async textToSpeech(params: {
    text: string;
    voiceId: string;
    stability?: number;
    similarityBoost?: number;
  }): Promise<Buffer> {

    const response = await axios.post(
      `${this.baseUrl}/text-to-speech/${params.voiceId}`,
      {
        text: params.text,
        model_id: 'eleven_monolingual_v1',
        voice_settings: {
          stability: params.stability || 0.5,
          similarity_boost: params.similarityBoost || 0.75
        }
      },
      {
        headers: {
```

```
        'xi-api-key': this.apiKey,
        'Content-Type': 'application/json'
      },
      responseType: 'arraybuffer'
    }
  );

  return Buffer.from(response.data);
}

async cloneVoice(params: {
  name: string;
  audioFiles: Buffer[];
  description?: string;
}): Promise<{ voiceId: string }> {

  const formData = new FormData();
  formData.append('name', params.name);

  if (params.description) {
    formData.append('description', params.description);
  }

  params.audioFiles.forEach((file, index) => {
    formData.append('files', file, `sample${index}.mp3`);
  });

  const response = await axios.post(
    `${this.baseUrl}/voices/add`,
    formData,
    {
      headers: {
        ...formData.getHeaders(),
        'xi-api-key': this.apiKey
      }
    }
  );

  return { voiceId: response.data.voice_id };
```

```
  }

  async deleteVoice(voiceId: string): Promise<void> {
    await axios.delete(`${this.baseUrl}/voices/${voiceId}`, {
      headers: { 'xi-api-key': this.apiKey }
    });
  }

  async speechToSpeech(params: {
    audioBuffer: Buffer;
    voiceId: string;
  }): Promise<Buffer> {

    const formData = new FormData();
    formData.append('audio', params.audioBuffer, 'input.mp3');
    formData.append('model_id', 'eleven_english_sts_v2');

    const response = await axios.post(
      `${this.baseUrl}/speech-to-speech/${params.voiceId}`,
      formData,
      {
        headers: {
          ...formData.getHeaders(),
          'xi-api-key': this.apiKey
        },
        responseType: 'arraybuffer'
      }
    );

    return Buffer.from(response.data);
  }
}

export const voiceService = new VoiceService();
```

---

5. QUEUE SERVICE (Job Processing)

TypeScript

```typescript
// services/queue.service.ts

import { Queue, Worker, Job } from 'bullmq';
import Redis from 'ioredis';
import { aiService } from './ai.service';
import { mediaService } from './media.service';
import { voiceService } from './voice.service';
import { db } from '@/lib/db';

const connection = new Redis({
  host: process.env.REDIS_HOST,
  port: parseInt(process.env.REDIS_PORT || '6379'),
  password: process.env.REDIS_PASSWORD,
  maxRetriesPerRequest: null
});

// ============================================
// VIDEO GENERATION QUEUE
// ============================================

export const videoQueue = new Queue('video-generation', {
connection });

export const videoWorker = new Worker(
  'video-generation',
  async (job: Job) => {
    const { userId, videoId, mode, params } = job.data;

    try {
      // 1. Update job status
      await db.processing_jobs.update(job.id, {
        status: 'processing',
        started_at: new Date(),
        current_step: 'Initializing'
      });

      await job.updateProgress(10);

      // 2. Generate audio if needed (Custom Mode)
```

```
      let audioUrl = params.audioUrl;

      if (mode === 'custom' && params.script &&
params.voiceId) {
        await job.updateProgress(20);
        await db.processing_jobs.update(job.id, {
          current_step: 'Generating audio...'
        });

        const audioBuffer = await voiceService.textToSpeech({
          text: params.script,
          voiceId: params.voiceId
        });

        const uploaded = await
mediaService.uploadAudio(userId, audioBuffer);
        audioUrl = uploaded.url;
      }

      await job.updateProgress(40);

      // 3. Generate video
      await db.processing_jobs.update(job.id, {
        current_step: 'Generating video frames...'
      });

      let aiJobId: string;

      if (mode === 'easy') {
        const result = await aiService.generateVideoEasyMode({
          imageUrl: params.imageUrl,
          script: params.script,
          actionPrompt: params.actionPrompt
        });
        aiJobId = result.jobId;
      } else {
        const result = await
aiService.generateVideoCustomMode({
          imageUrl: params.imageUrl,
```

```
          audioUrl: audioUrl!,
          actionPrompt: params.actionPrompt,
          duration: params.duration
        });
        aiJobId = result.jobId;
      }

      // 4. Poll for completion
      await job.updateProgress(60);
      await db.processing_jobs.update(job.id, {
        current_step: 'Rendering video...'
      });

      let videoUrl: string | null = null;
      let attempts = 0;
      const maxAttempts = 180; // 15 minutes (5s intervals)

      while (attempts < maxAttempts) {
        const status = await
aiService.checkJobStatus(aiJobId);

        if (status.status === 'completed') {
          videoUrl = status.output[0]; // Assuming URL
          break;
        }

        if (status.status === 'failed') {
          throw new Error(status.error || 'Video generation
failed');
        }

        // Update progress
        const progress = 60 + (status.progress || 0) * 0.35;
        await job.updateProgress(progress);

        await new Promise(resolve => setTimeout(resolve,
5000));
        attempts++;
      }
```

```
      if (!videoUrl) {
        throw new Error('Video generation timeout');
      }

      // 5. Download and re-upload to our storage
      await job.updateProgress(95);
      await db.processing_jobs.update(job.id, {
        current_step: 'Finalizing...'
      });

      const videoBuffer = await downloadFile(videoUrl);
      const uploaded = await mediaService.uploadVideo(userId,
videoBuffer, {
        filename: `video_${videoId}.mp4`,
        duration: params.duration || 8
      });

      // 6. Update database
      await db.videos.update(videoId, {
        url: uploaded.url,
        thumbnail_url: uploaded.thumbnailUrl,
        status: 'completed',
        completed_at: new Date(),
        processing_duration_seconds: Math.floor((Date.now() -
job.data.startTime) / 1000)
      });

      await db.processing_jobs.update(job.id, {
        status: 'completed',
        progress: 100,
        completed_at: new Date(),
        output_data: { videoUrl: uploaded.url }
      });

      await job.updateProgress(100);

      // 7. Notify user via WebSocket
      await notifyUser(userId, {
```

```
        type: 'video_completed',
        videoId,
        url: uploaded.url
      });

      return { success: true, videoUrl: uploaded.url };

    } catch (error) {
      // Handle error
      await db.videos.update(videoId, {
        status: 'failed',
        error_message: error.message
      });

      await db.processing_jobs.update(job.id, {
        status: 'failed',
        error_message: error.message
      });

      // Refund credits
      await creditsService.refund(userId, {
        type: 'video',
        amount: 1,
        reason: 'Generation failed'
      });

      throw error;
    }
  },
  {
    connection,
    concurrency: 5, // Process 5 videos concurrently
    limiter: {
      max: 10,
      duration: 60000 // Max 10 jobs per minute
    }
  }
);
```

```
// ==========================================
// IMAGE GENERATION QUEUE
// ==========================================

export const imageQueue = new Queue('image-generation', {
connection });

export const imageWorker = new Worker(
  'image-generation',
  async (job: Job) => {
    const { userId, imageId, type, params } = job.data;

    try {
      await job.updateProgress(10);

      let imageUrls: string[];

      if (type === 'generate') {
        imageUrls = await aiService.generateActor(params);
      } else if (type === 'edit') {
        imageUrls = await aiService.editImage(params);
      }

      await job.updateProgress(80);

      // Download and upload to our storage
      const uploadPromises = imageUrls!.map(async (url) => {
        const buffer = await downloadFile(url);
        return mediaService.uploadImage(userId, buffer, {
          filename: `image_${Date.now()}.png`,
          mimeType: 'image/png'
        });
      });

      const uploaded = await Promise.all(uploadPromises);

      // Save to database
      await db.images.createMany(
        uploaded.map(img => ({
```

```
          user_id: userId,
          url: img.url,
          thumbnail_url: img.thumbnailUrl,
          width: img.width,
          height: img.height,
          source: type,
          prompt: params.prompt
        }))
      );

      await job.updateProgress(100);

      return { success: true, images: uploaded };

    } catch (error) {
      // Refund credits
      await creditsService.refund(userId, {
        type: 'image',
        amount: 1,
        reason: 'Generation failed'
      });

      throw error;
    }
  },
  {
    connection,
    concurrency: 10
  }
);

// Helper function
async function downloadFile(url: string): Promise<Buffer> {
  const response = await axios.get(url, { responseType:
'arraybuffer' });
  return Buffer.from(response.data);
}
```

```
async function notifyUser(userId: string, data: any):
Promise<void> {
  // Implement WebSocket notification
  // socketService.emit(userId, data);
}
```

---

6. CREDITS SERVICE

TypeScript

```typescript
// services/credits.service.ts

class CreditsService {
  async deductCredits(userId: string, params: {
    type: 'video' | 'image' | 'voice';
    amount: number;
    description: string;
    relatedEntityId?: string;
  }): Promise<void> {

    const user = await db.users.findById(userId);

    // Check balance
    const currentBalance = params.type === 'video'
      ? user.video_credits_remaining
      : user.image_credits_remaining;

    if (currentBalance < params.amount) {
      throw new Error('Insufficient credits');
    }

    // Deduct
    const newBalance = currentBalance - params.amount;

    if (params.type === 'video') {
      await db.users.update(userId, {
        video_credits_remaining: newBalance
      });
```

```
    } else {
      await db.users.update(userId, {
        image_credits_remaining: newBalance
      });
    }

    // Record transaction
    await db.credit_transactions.create({
      user_id: userId,
      type: 'usage',
      amount: -params.amount,
      credit_type: params.type,
      description: params.description,
      balance_before: currentBalance,
      balance_after: newBalance,
      video_id: params.type === 'video' ?
params.relatedEntityId : null,
      image_id: params.type === 'image' ?
params.relatedEntityId : null
    });
  }

  async refund(userId: string, params: {
    type: 'video' | 'image';
    amount: number;
    reason: string;
  }): Promise<void> {

    const user = await db.users.findById(userId);

    const currentBalance = params.type === 'video'
      ? user.video_credits_remaining
      : user.image_credits_remaining;

    const newBalance = currentBalance + params.amount;

    if (params.type === 'video') {
      await db.users.update(userId, {
        video_credits_remaining: newBalance
```

```
      });
    } else {
      await db.users.update(userId, {
        image_credits_remaining: newBalance
      });
    }

    await db.credit_transactions.create({
      user_id: userId,
      type: 'refund',
      amount: params.amount,
      credit_type: params.type,
      description: `Refund: ${params.reason}`,
      balance_before: currentBalance,
      balance_after: newBalance
    });
  }

  async addCredits(userId: string, params: {
    videoCredits?: number;
    imageCredits?: number;
    source: 'purchase' | 'subscription_renewal' | 'bonus';
    description: string;
  }): Promise<void> {

    const user = await db.users.findById(userId);

    const updates: any = {};

    if (params.videoCredits) {
      updates.video_credits_remaining =
user.video_credits_remaining + params.videoCredits;

      await db.credit_transactions.create({
        user_id: userId,
        type: params.source,
        amount: params.videoCredits,
        credit_type: 'video',
        description: params.description,
```

```
        balance_before: user.video_credits_remaining,
        balance_after: updates.video_credits_remaining
      });
    }

    if (params.imageCredits) {
      updates.image_credits_remaining =
user.image_credits_remaining + params.imageCredits;

      await db.credit_transactions.create({
        user_id: userId,
        type: params.source,
        amount: params.imageCredits,
        credit_type: 'image',
        description: params.description,
        balance_before: user.image_credits_remaining,
        balance_after: updates.image_credits_remaining
      });
    }

    await db.users.update(userId, updates);
  }

  async resetMonthlyCredits(): Promise<void> {
    // Cron job to reset credits monthly
    const users = await db.users.findAll({
      where: {
        credits_reset_date: { lte: new Date() },
        subscription_status: 'active'
      }
    });

    for (const user of users) {
      const plan = await
db.subscription_plans.findByTier(user.subscription_tier);

      await this.addCredits(user.id, {
        videoCredits: plan.video_credits,
        imageCredits: plan.image_credits,
```

```
        source: 'subscription_renewal',
        description: 'Monthly credit renewal'
      });

      await db.users.update(user.id, {
        credits_reset_date: addMonths(new Date(), 1)
      });
    }
  }
}

export const creditsService = new CreditsService();
```

---

7. BILLING SERVICE (Stripe)

TypeScript

```
// services/billing.service.ts

import Stripe from 'stripe';
import { creditsService } from './credits.service';

class BillingService {
  private stripe: Stripe;

  constructor() {
    this.stripe = new Stripe(process.env.STRIPE_SECRET_KEY!, {
      apiVersion: '2023-10-16'
    });
  }

  async createCheckoutSession(params: {
    userId: string;
    planTier: string;
    billingCycle: 'monthly' | 'yearly';
    successUrl: string;
    cancelUrl: string;
  }): Promise<{ sessionId: string; url: string }> {
```

```
    const user = await db.users.findById(params.userId);
    const plan = await
db.subscription_plans.findByTier(params.planTier);

    // Get or create Stripe customer
    let customerId = user.stripe_customer_id;

    if (!customerId) {
      const customer = await this.stripe.customers.create({
        email: user.email,
        metadata: { userId: user.id }
      });
      customerId = customer.id;

      await db.users.update(user.id, {
        stripe_customer_id: customerId
      });
    }

    // Create checkout session
    const priceId = params.billingCycle === 'monthly'
      ? plan.stripe_price_id_monthly
      : plan.stripe_price_id_yearly;

    const session = await
this.stripe.checkout.sessions.create({
      customer: customerId,
      mode: 'subscription',
      payment_method_types: ['card'],
      line_items: [
        {
          price: priceId,
          quantity: 1
        }
      ],
      success_url: params.successUrl,
      cancel_url: params.cancelUrl,
      metadata: {
```

```
        userId: user.id,
        planTier: params.planTier
      }
    });

    return {
      sessionId: session.id,
      url: session.url!
    };
  }

  async createCustomerPortalSession(userId: string): Promise<{
url: string }> {
    const user = await db.users.findById(userId);

    if (!user.stripe_customer_id) {
      throw new Error('No Stripe customer found');
    }

    const session = await
this.stripe.billingPortal.sessions.create({
      customer: user.stripe_customer_id,
      return_url: `${process.env.APP_URL}/dashboard/billing`
    });

    return { url: session.url };
  }

  async handleWebhook(event: Stripe.Event): Promise<void> {
    switch (event.type) {
      case 'checkout.session.completed':
        await this.handleCheckoutCompleted(event.data.object
as Stripe.Checkout.Session);
        break;

      case 'customer.subscription.updated':
        await this.handleSubscriptionUpdated(event.data.object
as Stripe.Subscription);
        break;
```

```
      case 'customer.subscription.deleted':
        await
this.handleSubscriptionCanceled(event.data.object as
Stripe.Subscription);
        break;

      case 'invoice.payment_succeeded':
        await this.handlePaymentSucceeded(event.data.object as
Stripe.Invoice);
        break;

      case 'invoice.payment_failed':
        await this.handlePaymentFailed(event.data.object as
Stripe.Invoice);
        break;
    }
  }

  private async handleCheckoutCompleted(session:
Stripe.Checkout.Session): Promise<void> {
    const userId = session.metadata?.userId;
    if (!userId) return;

    const subscription = await
this.stripe.subscriptions.retrieve(
      session.subscription as string
    );

    const plan = await
db.subscription_plans.findByTier(session.metadata.planTier);

    // Update user
    await db.users.update(userId, {
      subscription_tier: plan.tier,
      subscription_status: 'active',
      subscription_id: subscription.id
    });
```

```
    // Add credits
    await creditsService.addCredits(userId, {
      videoCredits: plan.video_credits,
      imageCredits: plan.image_credits,
      source: 'subscription_renewal',
      description: `Subscription started: ${plan.name}`
    });

    // Record transaction
    await db.payment_transactions.create({
      user_id: userId,
      stripe_payment_intent_id: session.payment_intent as
string,
      amount_usd: (session.amount_total || 0) / 100,
      status: 'succeeded',
      transaction_type: 'subscription',
      video_credits_granted: plan.video_credits,
      image_credits_granted: plan.image_credits,
      description: `Subscription: ${plan.name}`
    });
  }

  private async handlePaymentSucceeded(invoice:
Stripe.Invoice): Promise<void> {
    // Monthly renewal
    const subscription = await
this.stripe.subscriptions.retrieve(
      invoice.subscription as string
    );

    const userId = subscription.metadata.userId;
    if (!userId) return;

    const user = await db.users.findById(userId);
    const plan = await
db.subscription_plans.findByTier(user.subscription_tier);

    // Add monthly credits
    await creditsService.addCredits(userId, {
```

```
      videoCredits: plan.video_credits,
      imageCredits: plan.image_credits,
      source: 'subscription_renewal',
      description: 'Monthly credit renewal'
    });
  }

  private async handleSubscriptionCanceled(subscription:
Stripe.Subscription): Promise<void> {
    const userId = subscription.metadata.userId;
    if (!userId) return;

    await db.users.update(userId, {
      subscription_status: 'cancelled',
      subscription_tier: 'free'
    });
  }
}

export const billingService = new BillingService();
```

🔄 REAL-TIME UPDATES (WebSocket)

TypeScript

```
// services/websocket.service.ts

import { Server as SocketIOServer } from 'socket.io';
import { Server as HTTPServer } from 'http';
import jwt from 'jsonwebtoken';

class WebSocketService {
  private io: SocketIOServer;
  private userSockets: Map<string, Set<string>> = new Map();

  initialize(httpServer: HTTPServer): void {
    this.io = new SocketIOServer(httpServer, {
      cors: {
```

```
        origin: process.env.FRONTEND_URL,
        credentials: true
      }
    });

    // Authentication middleware
    this.io.use(async (socket, next) => {
      const token = socket.handshake.auth.token;

      try {
        const decoded = jwt.verify(token,
process.env.JWT_SECRET!);
        socket.data.userId = decoded.userId;
        next();
      } catch (error) {
        next(new Error('Authentication error'));
      }
    });

    this.io.on('connection', (socket) => {
      const userId = socket.data.userId;

      // Track user connections
      if (!this.userSockets.has(userId)) {
        this.userSockets.set(userId, new Set());
      }
      this.userSockets.get(userId)!.add(socket.id);

      // Subscribe to job updates
      socket.on('subscribe:job', (jobId: string) => {
        socket.join(`job:${jobId}`);
      });

      socket.on('unsubscribe:job', (jobId: string) => {
        socket.leave(`job:${jobId}`);
      });

      socket.on('disconnect', () => {
        this.userSockets.get(userId)?.delete(socket.id);
```

```
      if (this.userSockets.get(userId)?.size === 0) {
        this.userSockets.delete(userId);
      }
    });
  });
}

// Send job progress update
emitJobProgress(jobId: string, data: {
  progress: number;
  status: string;
  currentStep?: string;
}): void {
  this.io.to(`job:${jobId}`).emit('job:progress', data);
}

// Send job completion
emitJobCompleted(jobId: string, data: any): void {
  this.io.to(`job:${jobId}`).emit('job:completed', data);
}

// Send job error
emitJobError(jobId: string, error: string): void {
  this.io.to(`job:${jobId}`).emit('job:error', { error });
}

// Send notification to user
emitToUser(userId: string, event: string, data: any): void {
  const sockets = this.userSockets.get(userId);
  if (sockets) {
    sockets.forEach(socketId => {
      this.io.to(socketId).emit(event, data);
    });
  }
}

// Broadcast to all users
broadcast(event: string, data: any): void {
  this.io.emit(event, data);
```

```
  }
}

export const websocketService = new WebSocketService();
```

🔐 MIDDLEWARE & SECURITY

TypeScript

```
// middleware/auth.middleware.ts

import { Request, Response, NextFunction } from 'express';
import jwt from 'jsonwebtoken';
import { db } from '@/lib/db';

export async function authenticate(
  req: Request,
  res: Response,
  next: NextFunction
): Promise<void> {
  try {
    const token = req.headers.authorization?.replace('Bearer
', '');

    if (!token) {
      res.status(401).json({ error: 'Authentication required'
});
      return;
    }

    const decoded = jwt.verify(token, process.env.JWT_SECRET!)
as { userId: string };
    const user = await db.users.findById(decoded.userId);

    if (!user) {
      res.status(401).json({ error: 'User not found' });
      return;
    }
```

```
    req.user = user;
    next();
  } catch (error) {
    res.status(401).json({ error: 'Invalid token' });
  }
}

// middleware/rate-limit.middleware.ts

import Redis from 'ioredis';

const redis = new Redis(process.env.REDIS_URL);

export function rateLimit(options: {
  maxRequests: number;
  windowMs: number;
  keyPrefix?: string;
}) {
  return async (req: Request, res: Response, next:
NextFunction) => {
    const userId = req.user?.id || req.ip;
    const key = `${options.keyPrefix ||
'rate-limit'}:${userId}`;

    const current = await redis.incr(key);

    if (current === 1) {
      await redis.expire(key, Math.floor(options.windowMs /
1000));
    }

    if (current > options.maxRequests) {
      res.status(429).json({
        error: 'Too many requests',
        retryAfter: await redis.ttl(key)
      });
      return;
    }
```

```
    res.setHeader('X-RateLimit-Limit', options.maxRequests);
    res.setHeader('X-RateLimit-Remaining', options.maxRequests
- current);

    next();
  };
}

// middleware/credits.middleware.ts

export function requireCredits(type: 'video' | 'image',
amount: number = 1) {
  return async (req: Request, res: Response, next:
NextFunction) => {
    const user = req.user!;

    const available = type === 'video'
      ? user.video_credits_remaining
      : user.image_credits_remaining;

    if (available < amount) {
      res.status(402).json({
        error: 'Insufficient credits',
        required: amount,
        available,
        upgradeUrl: '/dashboard/billing'
      });
      return;
    }

    next();
  };
}
```

📈 MONITORING & LOGGING

TypeScript

```typescript
// services/monitoring.service.ts

import * as Sentry from '@sentry/node';
import winston from 'winston';

// Initialize Sentry
Sentry.init({
  dsn: process.env.SENTRY_DSN,
  environment: process.env.NODE_ENV,
  tracesSampleRate: 1.0
});

// Winston logger
export const logger = winston.createLogger({
  level: process.env.LOG_LEVEL || 'info',
  format: winston.format.combine(
    winston.format.timestamp(),
    winston.format.errors({ stack: true }),
    winston.format.json()
  ),
  transports: [
    new winston.transports.File({ filename: 'error.log',
level: 'error' }),
    new winston.transports.File({ filename: 'combined.log' }),
    new winston.transports.Console({
      format: winston.format.simple()
    })
  ]
});

// Track metrics
export function trackMetric(name: string, value: number,
tags?: Record<string, string>): void {
  // Send to Datadog, CloudWatch, etc.
  logger.info('Metric', { name, value, tags });
}

// Track event
```

```
export function trackEvent(name: string, properties?:
Record<string, any>): void {
  logger.info('Event', { name, properties });
}
```

🚀 DEPLOYMENT & INFRASTRUCTURE

Docker Compose (Development):

```
None
version: '3.8'

services:
  app:
    build: .
    ports:
      - "3000:3000"
    environment:
      - NODE_ENV=development
      -
DATABASE_URL=postgresql://user:password@postgres:5432/speel
      - REDIS_URL=redis://redis:6379
    depends_on:
      - postgres
      - redis
    volumes:
      - .:/app
      - /app/node_modules

  postgres:
    image: postgres:15
    environment:
      POSTGRES_USER: user
      POSTGRES_PASSWORD: password
      POSTGRES_DB: speel
    ports:
      - "5432:5432"
```

```
    volumes:
      - postgres_data:/var/lib/postgresql/data

  redis:
    image: redis:7-alpine
    ports:
      - "6379:6379"
    volumes:
      - redis_data:/data

  worker:
    build: .
    command: npm run worker
    environment:
      - NODE_ENV=development
      -
DATABASE_URL=postgresql://user:password@postgres:5432/speel
      - REDIS_URL=redis://redis:6379
    depends_on:
      - postgres
      - redis
    volumes:
      - .:/app

volumes:
  postgres_data:
  redis_data:
```

Production Stack (Recommendation):

```
None
Frontend: Vercel
API/Backend: Railway o Fly.io
Database: Supabase PostgreSQL o AWS RDS
Redis: Upstash o AWS ElastiCache
Queue Workers: Railway (separate instances)
Storage: AWS S3 + CloudFront CDN
Monitoring: Sentry + Logtail
```

📋 ENVIRONMENT VARIABLES

```shell
# App
NODE_ENV=production
APP_URL=https://app.speel.co
FRONTEND_URL=https://speel.co
PORT=3000

# Database
DATABASE_URL=postgresql://user:password@host:5432/speel
DIRECT_URL=postgresql://user:password@host:5432/speel # For
Prisma migrations

# Redis
REDIS_HOST=your-redis-host
REDIS_PORT=6379
REDIS_PASSWORD=your-redis-password
REDIS_URL=redis://:password@host:6379

# Auth
JWT_SECRET=your-super-secret-jwt-key
JWT_EXPIRES_IN=7d
NEXTAUTH_SECRET=your-nextauth-secret
NEXTAUTH_URL=https://app.speel.co

# AWS S3
AWS_REGION=us-east-1
AWS_ACCESS_KEY_ID=your-access-key
AWS_SECRET_ACCESS_KEY=your-secret-key
S3_BUCKET=speel-media
CDN_URL=https://cdn.speel.co

# AI Services
REPLICATE_API_TOKEN=your-replicate-token
RUNPOD_API_KEY=your-runpod-key (if using)
SORA_API_KEY=your-sora-key (if available)
```

```
# 11 Labs
ELEVEN_LABS_API_KEY=your-elevenlabs-key

# Stripe
STRIPE_SECRET_KEY=sk_live_xxx
STRIPE_PUBLISHABLE_KEY=pk_live_xxx
STRIPE_WEBHOOK_SECRET=whsec_xxx

# Monitoring
SENTRY_DSN=https://xxx@sentry.io/xxx
LOGTAIL_SOURCE_TOKEN=your-logtail-token

# Email (SendGrid, Resend, etc.)
EMAIL_FROM=noreply@speel.co
SENDGRID_API_KEY=your-sendgrid-key

# Feature Flags
ENABLE_SORA=false
ENABLE_BATCH_GENERATION=true
MAX_VIDEO_DURATION=60
MAX_CONCURRENT_JOBS_PER_USER=3
```

✅ BACKEND DEVELOPMENT CHECKLIST

Phase 1: Core Setup (Week 1)

- Initialize Node.js/TypeScript project
- Setup PostgreSQL + Prisma
- Setup Redis
- Implement auth (JWT + NextAuth)
- Create database schema
- Setup API structure (REST/tRPC)

Phase 2: Media & Storage (Week 1)

- AWS S3 integration
- Image upload/processing (Sharp)
- Video upload/processing
- CDN setup (CloudFront)
- Thumbnail generation

Phase 3: AI Integration (Week 1)

- Replicate API client
- 11 Labs integration
- Video generation (Easy Mode)
- Video generation (Custom Mode)
- Image generation
- Image editing (Nano Banana equivalent)

Phase 4: Queue System (Week 1)

- BullMQ setup
- Video generation worker
- Image generation worker
- Progress tracking
- Error handling & retries
- Webhook handlers

Phase 5: Credits & Billing (Week 1)

- Credits system
- Stripe integration
- Subscription plans
- Checkout flow
- Webhook handlers
- Customer portal

Phase 6: Real-time (Week 2)

- WebSocket setup (Socket.io)
- Job progress events
- Notifications system
- User presence

Phase 7: Optimization (Week 2)

- Rate limiting
- Caching (Redis)
- Database indexing
- Query optimization
- API response compression

Phase 8: Monitoring (Week 2)

- Sentry error tracking
- Winston logging
- Metrics tracking
- Uptime monitoring

- Performance monitoring

Phase 9: Testing (Week 2)

- Unit tests
- Integration tests
- API endpoint tests
- Load testing
- Security audit

Phase 10: Deployment (Week 2)

- Docker setup
- CI/CD pipeline
- Production deployment
- Database migrations
- Monitoring setup
- Backup strategy