A.I. Film Director Assistant

Quandamooka Film Festival 2024/25

Part 1: Initial Overview

First, consider that the workflow involves multiple specialised agents (MoE), each focused on distinct phases and tasks, and a mix of multimodal AI models for text, imagery, audio, and even rough video sequences. A human facilitator (e.g., a creative director or community organizer) sits at the center, guiding and curating at key decision points. The aim is to scaffold a full pipeline: from initial story spark through to a polished pilot episode ready for local production and festival debut. Below is a possible end-to-end approach.

1. Ideation & Theme Setting

Agents Involved:

- **Cultural Context and Story Framing Agent:** Ingests localised cultural, historical, and environmental context (Quandamooka culture, Minjerribah settings, etc.) and assists in framing narrative options that resonate with local stories.
- **World-Building Agent:** Expands on initial thematic seeds, generating diverse storyworld pitches—e.g., a drama exploring island life transitions, an eco-thriller dealing with sea-level rise, a heartwarming comedy about cross-cultural friendships sparked by new visitors to the island film festival.

Human Involvement:
The local festival organiser or community storytellers pick a direction that best represents the community's voice and aspirations.

2. Character & Plot Development

Agents Involved:

- **Character Development Agent:** Crafts character backstories, relationships, and arcs that feel authentic to the community's identity. For example, it might generate a protagonist who is a young Quandamooka filmmaker returning home, or an elder who is a knowledge keeper guiding newcomers.
- **Plot Structuring Agent:** Lays out pilot episode plot outlines, scene-by-scene breakdowns, narrative pacing, and tension-building. It integrates local folklore, everyday struggles, humour, and environment-based story beats.

Human Involvement:
Local storytellers review character sketches and outlines, ensuring that cultural elements are respectfully represented and that the plot feels genuine.

3. Script Drafting & Dialogue Writing

Agents Involved:

- **Scriptwriting Agent (Dialogue Specialist):** Given the chosen plot structure, this agent generates a first full draft of the pilot script with scenes, stage directions, and dialogue that blend local language inflections, humour, and authenticity.
- **Editing & Consistency Agent:** Cross-checks for narrative coherence, character consistency, and cultural sensitivity. It might also ensure each character's voice is distinct, using a fine-tuned model that's "expert" in natural-sounding local speech patterns.

Human Involvement:
A local script consultant and community representatives read the draft, provide feedback on cultural tone, pacing, and authenticity. They can prompt the Agents to revise certain scenes, add subtle cultural references, or tone down clichés.

4. Visual Concepting & Storyboarding
Agents Involved:

- **Visual Concept Artist (Multi-Modal):** Uses text-to-image generation models (like Stable Diffusion variants or specialized fine-tuned image models) to create concept art of key locations (e.g., the island's iconic beaches, local bushland), character costumes, and mood boards.
- **Storyboard Agent:** Takes the script and concept art, generating rough storyboards or animatics (using image sequences and text-to-video models to give a moving preview). It might provide angle suggestions, lighting moods, and framing ideas for each scene.

Human Involvement:
Local filmmakers, cinematographers, or festival volunteers review the images and boards, selecting visuals that feel true to the island's look and feel, adjusting the style where needed.

5. Casting Assistance & Audition Scripts (Optional)
Agents Involved:

- **Character Voice & Casting Agent:** Suggests candidate actor profiles (fictional or hypothetical), voice tones, and even generates sample audition lines. Can produce audio snippets using text-to-speech.
- **Live-Action Feasibility Agent:** Evaluates the complexity of certain scenes for local production. Suggests how to simplify sets or props or adapt scenes to fit available community resources.

Human Involvement:
Festival organizers and local producers decide which characters can be played by community members, what props are manageable, and adapt as necessary.

6. Iterative Revisions & Polishing
Agents Involved:

- **Script Polisher (Style & Tone):** Refines wording, pacing, and dialogue after human feedback. Integrates local slang, tweaks emotional beats, and ensures pacing is festival-friendly.
- **Continuity & Quality Control Agent:** Scans the script and storyboard for inconsistencies, plot holes, or cultural missteps and suggests fixes.
- **Cultural Sensitivity Checker:** Specifically trained on cultural protocols and respectful storytelling, it flags any potential misrepresentations or inaccuracies.

Human Involvement:

The human team does a final read-through, possibly inviting community elders or cultural advisors to confirm the narrative honors local traditions.

7. Generating a Rough Animatic or Pre-Viz
Agents Involved:

- **Text-to-Video Prototype Agent:** Creates rough, stylized previews (not final production quality, but enough to visualize pacing and transitions).
- **Music & Sound Agent:** Suggests local-inspired soundtracks, ambient island sounds, or sample music tracks to match the emotional beats of scenes.

Human Involvement:
The community can comment on sound and visuals to ensure an authentic atmosphere—e.g., certain birdsong, the sound of waves at particular beaches, local instruments or music artists who might be featured.

8. Final Production Pack
Agents Involved:

- **Script Locking Agent:** Outputs a final shooting script with scene directions, dialogue, and notes for the production crew.
- **Style Guide Compiler:** Collates all relevant concept art, mood boards, sound references, and cultural guidelines into a neat production bible for local filmmakers.

Human Involvement:
The team approves the final pack, making it the single source of truth for filming.

9. On-Set Guidance (Human-led, AI-assisted)
During filming, the human director and crew shoot scenes based on the final script and production materials. They can use a lightweight on-site assistant (e.g., on a tablet) to ask for last-minute script clarifications or alternative line suggestions. But the main control is in human hands.

10. Post-Production Assistance
Agents Involved:

- **Rough Cut Editor Agent (Multi-Modal):** Suggests initial cuts of scenes based on best takes (assuming metadata from filming), continuity checks, and pacing guidelines.
- **Color Grading & Visual Effects Agent:** Suggests LUTs and color palettes that reflect the intended mood. Also can propose subtle VFX enhancements if wanted.
- **Dialogue Polishing & ADR Suggestions Agent:** Flags lines that might need re-recording for clarity, suggests improved intonation, or recommends background sound effects.

Human Involvement:
Local editors finalize cuts, using AI suggestions only as a baseline. Sound technicians layer in real local recordings. Human taste and craftsmanship guide final polish.

11. Final Quality Check & Community Screening
Agents Involved:

- **Quality Assurance Agent:** Performs a final run-through, checking for continuity, audio levels, and visual consistency.
- **Cultural Compliance Agent:** Ensures final product aligns with the festival's values, respects local traditions, and highlights positive storytelling aspects.

Human Involvement:
The community watches a rough final cut in a private screening and provides the last round of notes. Once approved, the pilot is ready for the public at the Quandamooka Film Festival.

Key Principles to Make it Work:

- Always keep a human in the loop at critical junctures: story selection, script approval, cultural checks, and final editing.
- Let specialized MoE agents handle their expert domains (script structure, dialogue, imagery, sound suggestions), but ensure their outputs are guided and validated by local knowledge.
- Use multi-modal generative AI for inspiration and rough visualization—real-world authenticity will come from community input and live filming.
- Flexibility: The workflow should allow easy iteration. The human crew can prompt agents to revise scenes, alter tone, or refine visuals at any stage.
- Local enrichment: Encourage the agents to incorporate local language patterns, cultural references, environmental details, and historical context so that the pilot feels uniquely "Minjerribah."

This approach can empower community members—especially those who've never made a film before—by giving them tools and guidance. It will also ensure that the final product is more than a generic production: it's enriched by local voices, local landscapes, and culturally aligned storytelling, all orchestrated by a seamless human-AI collaboration.

Part 2: Describing the Graphical User Interface

Below is a conceptual, screen-by-screen walkthrough of a GUI that aims to be intuitive, friendly, and accessible. The focus is on large icons, simple language, visual cues, and minimal jargon. The goal is to help anyone—regardless of age or tech-savviness—comfortably guide the process of creating a TV pilot episode, from story spark to final polish, assisted by the AI agents.

Overall Design Principles:

- **Visual Simplicity**: Large, colorful buttons and icons. Minimal text with plain language.

- **Step-by-Step Guidance**: Each stage presented as a “chapter” in the story-creation journey.
- **Cultural & Local Imagery**: Backgrounds featuring gentle illustrations inspired by Minjerribah’s beaches, local flora, and community icons.
- **Audio/Voice-Over Support**: Optional audio instructions for those who prefer listening over reading.
- **Progress Bar & Big Friendly Next/Back Buttons**: Users always know where they are and how to proceed.
- **Character Guide**: A friendly animated character (e.g., a friendly local animal spirit, like a cartoon dolphin or kookaburra) offering tips at each step.

1. Welcome Screen (Home)

Layout:

- A warm illustration of the island shore at sunrise.
- A big, friendly title: “Create Your Story!”
- A friendly animal guide character (e.g., “Koo the Kookaburra”) smiling and waving in the corner.
- **Three Big Buttons**:
  - **“Start New Story”** (a green button with a spark icon)
  - **“Continue Where I Left Off”** (a yellow button with a bookmark icon)
  - **“Explore Finished Stories”** (a blue button with a tiny TV icon)
- **Audio Option**: A small speaker icon in the top corner. Clicking it plays a brief friendly voice explaining what you can do on this page.

2. Ideation & Inspiration Page (“Let’s Imagine”)

Layout:

- A calm background of local bushland.
- At the top, a progress bar showing: **[Imagine] -> [Characters] -> [Script] -> [Visuals] -> [Final Steps]**. The current step “Imagine” is highlighted.
- **Main Content Area**:
  - **Headline**: “What’s Your Story About?”
  - A short sentence: “Think about what you want your show to feel like. Funny? Exciting? Maybe something that honors your home and traditions.”

- **Cultural Inspiration Carousel**: Photos and simple icon-based story prompts related to local traditions, nature, community life. Users can click left or right arrows to browse ideas.

- **Buttons**:
  - **"Use This Idea!"** once you pick a concept or prompt from the carousel.
  - **"More Ideas"** to see additional suggestions from the AI's Cultural Context Agent.
- Koo the Kookaburra hovers at the bottom-right, offering a tip in a speech bubble: "Pick something that makes you smile or feel proud!"

3. Characters Page ("Meet Your Friends & Heroes")

Layout:

- Background: A gentle watercolor illustration of a community gathering.
- Title: "Who are the people in your story?"
- **Character Cards**: Large, colorful cards each showing a character's name, a simple illustration, and a short description. Example:
  - **Aunty Moira**: An elder who knows the island's stories.
  - **Jai**: A teenager learning traditional arts.
  - **Marina**: A visitor from afar who wants to understand local customs.
- Users can click **"Add a Character"** to prompt the Character Development Agent for new suggestions or **"Edit Character"** to tweak a personality trait (through a simple slider or yes/no questions).
- A "Cultural Sensitivity Check" button appears as a small badge on each character. Clicking it reveals a friendly message: "This character feels respectful and authentic. Would you like to adjust anything?"
- **Navigation Buttons**:
  - **Back to Ideation**
  - **Next: Build the Story Scenes**

Koo appears: "Great job! These characters look wonderful. Ready to see what they do next?"

4. Plot & Scene Outline Page ("Build Your Story Steps")

Layout:

- Background: A simple line-drawn map of the island, with small icons marking possible story locations.
- Title: "What Happens First, Next, and Last?"

- **Scene Timeline**: A horizontal row of big numbered circles (Scene 1, Scene 2, Scene 3…). Each scene expands when clicked, showing a short description.
- Users can drag and drop scenes to reorder them (with big, easy-to-grab arrows), or press a "Shuffle Ideas" button to get different plot suggestions.
- Simple language buttons like **"Make it Funnier"**, **"Make it More Adventurous"**, or **"Slow Down, More Details"** help refine scenes without technical jargon.
- **Next** button leads to "Write Dialogue".

Koo appears: "Your story is coming together! Pick the order of events that feels best. If you're unsure, try 'Shuffle Ideas'."

5. Script Writing Page ("Let's Talk!")

Layout:

- Background: A simple indoor living room setting (imagine a cozy community hall).
- Title: "Give Your Characters Something to Say"
- On the left: A column listing scenes. When a scene is selected, the center area shows script dialogue suggestions line by line with character names.
- Users can click **"Hear This Scene"** to have the script read aloud.
- Users can tweak lines by choosing from simple alternatives: "More caring tone," "Funnier line," "Shorter sentence."
- A **Cultural Check** icon appears if users want to ensure dialogue respects local language patterns.
- **Save & Continue** button moves on after humans are happy with the script.

Koo: "Your characters have a voice! Pick words that sound right. Need help? Try 'More Caring Tone' for gentler dialogue."

6. Visual Concepts & Storyboards ("Picture It!")

Layout:

- Background: A gallery wall.
- Title: "See Your Story Come to Life"
- **Concept Art Thumbnails**: Colorful illustrations of key locations, characters, and scenes. Clicking a thumbnail opens a larger view.
- Buttons like **"More Local Style"** (the system adjusts images to look more authentic to local aesthetics), **"Change Mood"** (brighter colors, softer lighting), or **"Simplify This Scene"** are always in plain language.

- Storyboard: A left-right scroll of frames, each frame represented by a simple cartoon panel. Click any frame to adjust mood, add notes, or get new suggestions.
- **Cultural Sensitivity Badge**: A small green leaf icon on each image that can be clicked to confirm authenticity. If something's off, suggestions appear in easy-to-read text.

Koo: "Looks great! Want to brighten that beach scene or add more local flowers? Give it a try!"

7. Casting & Practical Choices ("Who Will Play the Parts?")

Layout:

- Background: A community hall with friendly silhouettes.
- Title: "Pick Actors and Plan It Out"
- Suggestion cards: Each card shows a hypothetical actor profile (e.g., "Local Teenager," "Respected Elder," "Visitor with Musical Talent").
- Users can click "Simple Props Only" or "Shorter Scenes" to adapt the production to local resources.
- A big **"Approve Plan"** button when ready.

Koo: "Choose who might play these roles. If you don't know exact people, think about who in the community fits best!"

8. Final Script & Production Pack ("Ready to Roll!")

Layout:

- Background: A neat binder with colorful tabs.
- Title: "Your Story's Blueprint"
- A summary screen showing the final script (scrollable, with large text), selected concept art, and a "Production Checklist" with big, friendly checkmarks.
- Buttons: **"Print Friendly Version"**, **"Download Story Pack"**, and **"Make Minor Changes"** if needed.

Koo: "Everything is set! Your script, images, and notes are ready. Want to print or just keep it here?"

9. Pre-Visualization ("See a Rough Preview")

Layout:

- Background: A small movie screen icon.
- A simple play button to watch a rough animatic (low-quality but understandable).
- Buttons: **"More Lively Music"**, **"Quieter Background"**, **"Slower Pace."**

- After adjustments, a big “I Like It!” button confirms approval.

Koo: “Let’s watch a little preview. It’s not final, but it’ll give you a feel for how it looks and sounds!”

10. Post-Production Assistance & Community Review (“Polish It Up!”)

Layout:

- Background: A friendly workshop setting with paintbrushes and microphones.
- Title: “Make It Shine”
- Users see a final cut preview with big Pause/Play buttons and volume controls.
- Simple sliders for brightness, sound clarity, and captions on/off.
- Community Review: A big “Show to Friends” button lets you record reactions or gather feedback in a fun, kid-friendly poll format (e.g., “Thumbs Up,” “Needs More Fun,” “Change Music”).
- After feedback: Buttons like **“Apply Suggestions”** or **“I’ll Edit Myself”**.

Koo: “Your story is almost perfect! If anyone says it needs more brightness or a clearer voice, just use these easy sliders.”

11. Final Celebration Screen (“You Did It!”)

Layout:

- Background: Fireworks over the island’s horizon.
- Title: “Your Story Is Ready!”
- A simple congratulatory message: “Congratulations! Your pilot is complete and ready for the festival!”
- Buttons: **“Share With Festival”**, **“Start a New Story”**, or **“Exit”**.
- Koo: “Well done! Your story is ready to bring people together and celebrate our community. Want to make another one?”

Additional Accessibility Features:

- **Language Toggle**: A dropdown at the top (with big flags/icons) to switch language if needed.
- **Text Size Adjuster**: A large A and a small A icon in the corner to make text bigger or smaller.
- **Audio Guidance**: The speaker icon on each page to read the instructions aloud.
- **Color-Blind Friendly Palettes** and High-Contrast Mode: A toggle in the corner for better visibility.

End Result:
The GUI is a friendly step-by-step story creation "adventure," with minimal technical language, abundant visuals, easy adjustments, and supportive cultural checks. Even children or elderly with no filmmaking or tech background should feel comfortable exploring, adjusting, and finalizing their pilot episode for the Quandamooka Film Festival.

Part 3: Human Readable Pseudo-Code

Below is a structured pseudo code outline that can be interpreted by non-coding humans and AI agents alike, offering a clear step-by-step workflow. Think of it as a reference "script" that both people and AI modules can follow or check against to ensure clarity, authenticity, and coherence. Each step describes what should happen, who does what, and how to confirm quality before moving on.

Global Variables and States:

```
DEFINE Global Variables:
    Project_State = "IDEATION"   // Current stage of the workflow
    Human_Approval = FALSE       // Tracks if human overseers have approved the
current output
    Cultural_Advisor_Input = "PENDING"  // Collects feedback from local cultural
advisors
    Revision_Count = 0           // How many times revisions have been made this
cycle
    MAX_REVISIONS = 5            // Reasonable cap on revisions to prevent endless
loops
    All_Agents = [Cultural_Context_Agent, World_Building_Agent,
Character_Dev_Agent,
                  Plot_Structure_Agent, Scriptwriting_Agent, Editing_Agent,
                  Visual_Concept_Agent, Storyboard_Agent, Casting_Assistant,
                  Continuity_Checker, Cultural_Sensitivity_Checker,
                  Text_to_Video_Proto_Agent, Sound_Agent, Final_QA_Agent]

    // Data structures to hold outputs:
    Narrative_Concepts = []
    Selected_Concept = ""
    Character_Profiles = []
    Scene_Outlines = []
    Draft_Script = ""
    Concept_Art = []
    Storyboards = []
    Final_Script = ""
    Production_Pack = {}
    Rough_Cut = {}
    Final_Product = {}
```

Workflow Steps:

Step 1: Ideation Phase

```
Project_State = "IDEATION"
CALL Cultural_Context_Agent WITH [local histories, cultural facts, environment
data]
    OUTPUT -> Narrative_Concepts  // e.g., multiple story pitches
```

```
DISPLAY Narrative_Concepts TO Human_Overseers
WAIT FOR Human Selection:
    SELECTED = User_Choice_of_Concept // Human picks best concept
IF SELECTED IS NOT NULL:
    Selected_Concept = SELECTED
    Human_Approval = TRUE
ELSE:
    REQUEST another round from Cultural_Context_Agent
    // Loop until concept is chosen or project is terminated
```

Quality/Confidence Check:

- Ensure that narrative concepts respect cultural heritage:

```
CALL Cultural_Sensitivity_Checker WITH Narrative_Concepts
IF Cultural_Sensitivity_Checker FLAGS issues:
    REVISE concepts via Cultural_Context_Agent
ELSE:
    CONTINUE
```

Step 2: World-Building & Plot Structuring

```
IF Human_Approval == TRUE AND Project_State == "IDEATION":
    Project_State = "STORY_DEVELOPMENT"

CALL World_Building_Agent WITH Selected_Concept
    OUTPUT -> Extended Storyworld Background

CALL Plot_Structure_Agent WITH Extended Storyworld Background
    OUTPUT -> Scene_Outlines (Act-by-Act breakdown)
```

Human Involvement:

```
DISPLAY Scene_Outlines TO Human_Overseers AND Cultural_Advisors
WAIT FOR FEEDBACK:
    IF Feedback_Indicates_Changes:
        Revision_Count += 1
        CALL Plot_Structure_Agent TO REVISE Scene_Outlines
        IF Revision_Count > MAX_REVISIONS:
            REQUEST Manual Human Overhaul or Proceed with Compromise
    ELSE:
        Human_Approval = TRUE
```

Quality/Confidence Check:

- Ensure coherence and narrative logic:

```
CALL Continuity_Checker WITH Scene_Outlines
IF Continuity_Checker DETECTS GAPS:
    Scene_Outlines = REVISE VIA Plot_Structure_Agent
    Repeat until no major gaps or Revision_Count > MAX_REVISIONS
```

Step 3: Character Development

```
IF Human_Approval == TRUE AND Project_State == "STORY_DEVELOPMENT":
    Project_State = "CHARACTER_DEVELOPMENT"

CALL Character_Dev_Agent WITH [Selected_Concept, Scene_Outlines]
    OUTPUT -> Character_Profiles (heroes, supporting roles, unique traits)
```

Human Involvement:

```
DISPLAY Character_Profiles TO Human_Overseers & Cultural_Advisors
WAIT FOR FEEDBACK:
    IF Characters Need More Authenticity:
        Revision_Count += 1
        CALL Character_Dev_Agent TO REFINE Character_Profiles
        IF Revision_Count > MAX_REVISIONS:
            Human_Overseers MANUALLY EDIT CHARACTERS
    ELSE:
        Human_Approval = TRUE
```

Quality Check:

- Check cultural accuracy and depth:

```
CALL Cultural_Sensitivity_Checker WITH Character_Profiles
IF FLAGGED:
    REVISE via Character_Dev_Agent
```

Step 4: Script Drafting

```
IF Human_Approval == TRUE AND Project_State == "CHARACTER_DEVELOPMENT":
    Project_State = "SCRIPT_DRAFTING"

CALL Scriptwriting_Agent WITH [Scene_Outlines, Character_Profiles]
    OUTPUT -> Draft_Script (full pilot draft)
```

Human Involvement:

```
DISPLAY Draft_Script TO Human_Overseers, Cultural_Advisors, Local Storytellers
WAIT FOR FEEDBACK:
    IF Script Needs Changes (tone, pacing, language, cultural details):
        Revision_Count += 1
        CALL Editing_Agent TO REFINE Draft_Script
        IF Revision_Count > MAX_REVISIONS:
            Human_Overseers MANUALLY EDIT SECTIONS
    ELSE:
        Human_Approval = TRUE
```

Quality Check:

- Check for narrative flow and authenticity:

```
CALL Continuity_Checker WITH Draft_Script
CALL Cultural_Sensitivity_Checker WITH Draft_Script
IF ANY ISSUES DETECTED:
    Revision_Count += 1
    CALL Editing_Agent TO CORRECT
    REPEAT UNTIL CLEARED OR MAX_REVISIONS REACHED

---

### Step 5: Visual Concepts & Storyboarding
```

IF Human_Approval == TRUE AND Project_State == “SCRIPT_DRAFTING”:
Project_State = “VISUAL_DEVELOPMENT”

```
CALL Visual_Concept_Agent WITH Draft_Script
OUTPUT -> Concept_Art (key locations, moods, character looks)

CALL Storyboard_Agent WITH Draft_Script & Concept_Art
OUTPUT -> Storyboards (scene-by-scene visual frames)

**Human Involvement**:

DISPLAY Concept_Art, Storyboards TO Human_Overseers & Community Artists
WAIT FOR FEEDBACK:
IF Visuals Are Inauthentic or Need Tweaks:
Revision_Count += 1
CALL Visual_Concept_Agent AND/OR Storyboard_Agent TO ADJUST
IF Revision_Count > MAX_REVISIONS:
HUMAN ARTISTS MAKE FINAL DECISIONS
ELSE:
Human_Approval = TRUE

---

### Step 6: Feasibility & Casting Suggestions

IF Human_Approval == TRUE AND Project_State == “VISUAL_DEVELOPMENT”:
Project_State = “PRODUCTION_PREP”

CALL Casting_Assistant WITH Character_Profiles
OUTPUT -> Suggested Casting Profiles, Voice Tones, Audition Lines

CALL Live-Action_Feasibility_Agent WITH Storyboards
OUTPUT -> Production_Simplification_Suggestions (adapt scene complexity to local resources)

**Human Involvement**:

DISPLAY Casting Suggestions & Feasibility Tips TO Production Team
WAIT FOR FEEDBACK:
IF Suggestions Not Viable:
Revision_Count += 1
CALL Casting_Assistant / Feasibility_Agent TO ADAPT
IF Revision_Count > MAX_REVISIONS:
HUMAN PRODUCTION TEAM DECIDES MANUALLY
ELSE:
Human_Approval = TRUE

---

### Step 7: Final Script & Production Pack

IF Human_Approval == TRUE AND Project_State == “PRODUCTION_PREP”:
Project_State = “SCRIPT_LOCK”
```

```
CALL Script_Polisher WITH Draft_Script
OUTPUT -> Final_Script (clean, locked version)

COLLECT Concept_Art, Storyboards, Casting Notes, Cultural Guidelines, Scene Breakdown
STORE INTO Production_Pack
```

Human Involvement:

```
DISPLAY Final_Script & Production_Pack TO Whole Creative Team
WAIT FOR SIGN-OFF FROM Cultural_Advisors & Producers:
IF Changes Needed:
Revision_Count += 1
CALL Script_Polisher AGAIN
IF Revision_Count > MAX_REVISIONS:
HUMAN TEAM FINALIZES MANUALLY
ELSE:
Human_Approval = TRUE
```

Step 8: Rough Pre-Visualization

```
IF Human_Approval == TRUE AND Project_State == "SCRIPT_LOCK":
Project_State = "PRE_VISUALIZATION"

CALL Text_to_Video_Proto_Agent WITH Final_Script & Storyboards
OUTPUT -> Rough_Animatic or Pre-Vis Video

CALL Sound_Agent WITH Scene_Outlines
OUTPUT -> Suggested Temp Music, Ambient Sounds
```

Human Involvement:

```
DISPLAY Rough_Animatic & Sounds TO Human_Overseers & Community Reviewers
WAIT FOR FEEDBACK:
IF Visual or Audio Doesn't Match Cultural Aesthetic:
Revision_Count += 1
CALL Text_to_Video_Proto_Agent / Sound_Agent TO ADJUST
IF Revision_Count > MAX_REVISIONS:
HUMAN TEAM MANUALLY SELECT SOUND/VIDEO REFERENCES
ELSE:
Human_Approval = TRUE
```

Step 9: Production & Filming (Human-Led)

```
IF Human_Approval == TRUE AND Project_State == "PRE_VISUALIZATION":
Project_State = "PRODUCTION_PHASE"

HUMAN CREW FILMS SCENES FOLLOWING Production_Pack
```

```
IF ON-SET ISSUES ARISE:
    HUMAN CREW CONSULT Agents (e.g., Continuity_Checker) FOR QUICK SOLUTIONS

---

### Step 10: Post-Production Assistance

Project_State = "POST_PRODUCTION"

CALL Rough_Cut_Editor_Agent WITH Filmed_Footage
OUTPUT -> Initial Rough_Cut

CALL Continuity_Checker WITH Rough_Cut
IF ISSUES FOUND:
    CALL Rough_Cut_Editor_Agent TO FIX EDITS

CALL Color_Grading_Agent WITH Rough_Cut
OUTPUT -> Suggest Color Palettes

CALL Sound_Agent WITH Rough_Cut
OUTPUT -> Improved Audio Mix, ADR Suggestions

**Human Involvement**:

DISPLAY Rough_Cut, Color Suggestions, Audio Mix TO Editing Team & Cultural Advisors
WAIT FOR FEEDBACK:
    IF Changes Needed:
        Revision_Count += 1
        CALL Agents accordingly (Editor, Sound, Color)
        IF Revision_Count > MAX_REVISIONS:
            HUMAN EDITORS TAKE MANUAL CONTROL
    ELSE:
        Human_Approval = TRUE

---

### Step 11: Final Quality Check & Approval

IF Human_Approval == TRUE AND Project_State == "POST_PRODUCTION":
    Project_State = "FINAL_REVIEW"

CALL Final_QA_Agent WITH Final_Product (locked edit)
CHECK FOR Continuity, Audio Levels, Visual Consistency

CALL Cultural_Sensitivity_Checker WITH Final_Product
IF ANY ISSUES:
    MAKE NECESSARY EDITS (HUMAN OR AGENT-ASSISTED)

**Human Involvement**:

DISPLAY Final_Product TO Entire Community Panel
```

```
WAIT FOR FINAL SIGN-OFF:
IF APPROVED:
Human_Approval = TRUE
ELSE:
ADDRESS FEEDBACK UNTIL ACCEPTABLE
```

```
---

**If at any point the Humans or Community Advisors are unsatisfied and no further
automated revisions can solve the problem, the process defaults to human-driven
editing. The pseudo code ensures a transparent, trackable loop: propose -> review -
> revise -> approve.**

**End State**: Once Human_Approval == TRUE at the FINAL_REVIEW stage, the pilot is
ready for the Quandamooka Film Festival.
```

Part 4: High-Level Code Development

Below is a high-level architecture and example code snippets to implement the described system as a website. The approach uses a common modern stack: a React-based frontend for the GUI, a Node.js/Express backend for orchestrating AI requests, and possibly a Python-based microservice layer (if integrating large language models or multimodal models via APIs). These examples are illustrative, not production-ready, but give a solid starting point.

System Architecture

Frontend (React/Next.js):

- Responsible for rendering the GUI with large icons, step-by-step wizards, cultural sensitivity checks, sliders, etc.
- Communicates with a backend API for operations such as generating story concepts, characters, scripts, images, and previews.
- Stores local state (current step, chosen concept, etc.) and updates the UI as responses come in.

Backend (Node.js/Express):

- Acts as a controller that routes requests between the frontend and various AI services.
- Maintains session state (e.g., user choices, project progress) in a database or in-memory store.
- Calls out to specialized AI agent endpoints (possibly running as separate services) to fetch data.
- Implements caching, rate-limiting, and input validation.

AI Services (Python/FastAPI or Flask):

- Multiple specialized endpoints for each agent (e.g., /cultural_context, /plot_structure, /scriptwriter, /cultural_sensitivity_check).
- Uses prompt templates and model calls (e.g., to OpenAI, Anthropic, or local LLMs) to produce results.
- Returns JSON responses to the Node.js backend.

Database (e.g., PostgreSQL or MongoDB):

- Stores user sessions, chosen concepts, script drafts, and metadata for each project.
- Ensures that if users return later, they can "Continue Where I Left Off."

Frontend Components (Example using React)

Key Dependencies:

- React or Next.js for SSR
- TailwindCSS or MUI for accessible, large, colorful UI components
- Axios or Fetch API for making backend requests

Example: App.js

```
import React, { useState, useEffect } from 'react';
import { BrowserRouter as Router, Route, Switch, Link } from 'react-router-dom';
import Home from './pages/Home';
import Ideation from './pages/Ideation';
import Characters from './pages/Characters';
import ScriptDrafting from './pages/ScriptDrafting';
// ... other pages

function App() {
  return (
    <Router>
      <div className="font-sans bg-gray-100 min-h-screen">
        <Switch>
          <Route path="/" exact component={Home} />
          <Route path="/ideation" component={Ideation} />
          <Route path="/characters" component={Characters} />
          <Route path="/script" component={ScriptDrafting} />
          {/* Add routes for Visual, Casting, etc. */}
        </Switch>
      </div>
    </Router>
  );
}

export default App;
```

Example: pages/Home.js

```
import React from 'react';
import { Link } from 'react-router-dom';

export default function Home() {
  return (
    <div className="flex flex-col items-center justify-center p-10">
      <h1 className="text-4xl font-bold mb-4">Create Your Story!</h1>
      <div className="flex flex-col gap-4">
        <Link to="/ideation" className="bg-green-500 px-6 py-3 text-white rounded-
lg text-2xl">Start New Story</Link>
        <Link to="/resume" className="bg-yellow-500 px-6 py-3 text-white rounded-lg
text-2xl">Continue Where I Left Off</Link>
        <Link to="/gallery" className="bg-blue-500 px-6 py-3 text-white rounded-lg
text-2xl">Explore Finished Stories</Link>
      </div>
      {/* Optionally add a toggle for audio instructions */}
    </div>
  );
}
```

Example: pages/Ideation.js

```
import React, { useState, useEffect } from 'react';
import axios from 'axios';

export default function Ideation() {
  const [concepts, setConcepts] = useState([]);
  const [selectedConcept, setSelectedConcept] = useState(null);

  useEffect(() => {
    axios.get('/api/concepts').then(res => {
      setConcepts(res.data);
    });
  }, []);

  const handleSelectConcept = (concept) => {
    setSelectedConcept(concept);
    // Post selection to backend to store state
    axios.post('/api/select-concept', { concept });
  };

  return (
    <div className="p-8">
      <h2 className="text-3xl font-semibold mb-4">What’s Your Story About?</h2>
      <div className="flex overflow-x-scroll space-x-4">
        {concepts.map((c, idx) => (
          <div key={idx} className="bg-white p-4 rounded-lg shadow min-w-[200px]">
            <p>{c.description}</p>
            <button
              onClick={() => handleSelectConcept(c)}
              className="mt-2 bg-green-500 text-white px-4 py-2 rounded">
              Use This Idea!
            </button>
          </div>
        ))}
      </div>
      {selectedConcept && (
        <div className="mt-6">
          <p className="text-xl">You selected: {selectedConcept.title}</p>
          <a href="/characters" className="bg-blue-500 text-white px-4 py-2 rounded
inline-block mt-4">Next: Characters</a>
        </div>
      )}
    </div>
  );
}
```

Backend (Node.js/Express) Setup

Key Steps:

- Implement endpoints like /api/concepts to return story ideas.
- Implement POST endpoints like /api/select-concept to store the user's choice in a session.
- Communicate with AI services by calling Python endpoints or OpenAI's API directly from the Node server.

Example: server.js (Node/Express)

```
const express = require('express');
const cors = require('cors');
const session = require('express-session');
const axios = require('axios');

const app = express();
app.use(cors());
app.use(express.json());

app.use(session({
  secret: 'some_secret_key',
  resave: false,
  saveUninitialized: true
}));

// Mock concepts endpoint - in practice, you'd call Cultural_Context_Agent service.
app.get('/api/concepts', async (req, res) => {
  // Example call to AI microservice
  const response = await axios.post('http://localhost:8000/cultural_context', {
    locale: 'Minjerribah',
    theme: 'community'
  });
  // response.data might be { concepts: [...] }
  res.json(response.data.concepts);
});

app.post('/api/select-concept', (req, res) => {
  req.session.selectedConcept = req.body.concept;
  res.json({status: 'ok'});
});

// Other endpoints for character generation, script drafting, etc.

const PORT = process.env.PORT || 3001;
app.listen(PORT, () => console.log(`Backend running on port ${PORT}`));
```

AI Services (Python/FastAPI Example)

Example: cultural_context_agent.py

```
from fastapi import FastAPI
import uvicorn
```

```
from typing import Dict

app = FastAPI()

@app.post("/cultural_context")
async def cultural_context(data: Dict):
    locale = data.get('locale', 'Minjerribah')
    theme = data.get('theme', 'community')
    # In reality, call an LLM to generate concepts
    # For demonstration:
    concepts = [
        {"title": "Island Sunrise Story", "description": "A gentle tale of a family welcoming the new day."},
        {"title": "Festival of Voices", "description": "A community gathering celebrates music and tradition."},
        {"title": "Tides of Change", "description": "An exploration of shifting cultures as visitors arrive."}
    ]
    return {"concepts": concepts}

if __name__ == "__main__":
    uvicorn.run(app, host="0.0.0.0", port=8000)
```

Similarly, create endpoints for plot structure, character dev, etc. Each would use a prompt sent to a model (like OpenAI's API via openai Python library) and return the structured output.

Integration Steps

1. **Front to Back**: The React app (frontend) calls the Node.js API. The Node.js API calls Python microservices for AI tasks.
2. **Session Handling**: User selections (chosen concept, characters, etc.) are stored in req.session or a database, so refreshing the page doesn't lose progress.
3. **Styling & Accessibility**: Use large fonts, contrast, and icons. Provide a toggle for voice guidance.
4. **Deployment**:
   - Run Python services on separate servers (or containers) exposed on internal network.
   - Node.js as the main backend API layer.
   - React/Next.js static files served via a CDN or Node.js server.
   - Use HTTPS and proper security measures.

Additional Features

- **Audio Guidance**: Include a text-to-speech feature on the frontend. For example, integrate with Web Speech API or a text-to-speech API to read out instructions.
- **Localization & Language Toggles**: A simple state-managed language switch could update all text from a translation file.
- **Database Integration**: Implement database logic in the Node.js layer to store user progress:

```
// Example using PostgreSQL
// On concept select:
await db.query('INSERT INTO user_choices (session_id, concept) VALUES ($1, $2)',
[req.sessionID, req.body.concept]);
```

End Result:
By following this approach—React for the GUI, Node.js for API routing and session management, Python microservices for LLM and multimodal operations—you can build a fully functioning website that is friendly for kids and elders, orchestrating the entire TV pilot creation workflow with human-friendly interfaces and robust AI-powered backends.

Part 5: Delving Deeper into the Code

Below is an expanded version of the system's codebase, focusing on more production-readiness. This includes environment-based configuration, better error handling, logging, database integration, linting/code style recommendations, and a rough directory structure. The examples remain illustrative rather than fully turnkey, but they outline the steps and components you'd typically implement when moving from a prototype to a production-ready system.

Refined Architecture & Directory Structure

Directory Structure Example:

```
project-root/
  frontend/
    src/
      pages/
      components/
      styles/
      ...
    package.json
    ... (React build configs)

  backend/
    src/
      config/
        config.js
      db/
        migrations/
        seeds/
        schema.sql
      middlewares/
        errorHandler.js
      routes/
        concepts.js
        characters.js
        script.js
        index.js
      services/
        aiClient.js
        sessionService.js
      utils/
        logger.js
        validators.js
    package.json
    .env
    ...

  ai-services/
    cultural_context_agent/
      app.py
      requirements.txt
      ...
    plot_structure_agent/
      ...
    scriptwriting_agent/
      ...

  docker/
    Dockerfile.backend
    Dockerfile.frontend
    docker-compose.yml

  docs/
    README.md
    ARCHITECTURE.md
    DEPLOYMENT.md
```

This structure separates concerns:

- **frontend/**: Contains the React/Next.js application.
- **backend/**: Node.js/Express server code with routes, services, DB migrations, and config files.
- **ai-services/**: Python microservices.
- **docker/**: Dockerfiles and docker-compose for containerization.
- **docs/**: Documentation.

Environment Variables & Configuration

Use .env files and a config module to manage environment variables. In backend/src/config/config.js:

```
require('dotenv').config();

module.exports = {
  port: process.env.PORT || 3001,
  databaseUrl: process.env.DATABASE_URL,
  aiServices: {
    culturalContextUrl: process.env.CULTURAL_CONTEXT_URL ||
'http://localhost:8000/cultural_context',
    // Add URLs for other AI services
  },
  sessionSecret: process.env.SESSION_SECRET || 'some_strong_secret',
  logLevel: process.env.LOG_LEVEL || 'info'
};
```

.env file example:

```
PORT=3001
DATABASE_URL=postgresql://user:password@db:5432/mydatabase
CULTURAL_CONTEXT_URL=http://cultural_context_agent:8000/cultural_context
SESSION_SECRET=super_secret_session_key
LOG_LEVEL=info
```

In production, you'd inject these as environment variables into your container platform (e.g., Kubernetes secrets, AWS ECS Secrets, etc.).

Database Integration

Use a relational database (e.g., PostgreSQL) and run migrations. In backend/src/db/schema.sql you might have:

```sql
CREATE TABLE IF NOT EXISTS user_sessions (
    id SERIAL PRIMARY KEY,
    session_id VARCHAR(255) UNIQUE NOT NULL,
    selected_concept JSONB,
    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP DEFAULT NOW()
);

-- Add tables for characters, script drafts, etc. as needed
```

Run migrations using a tool like knex or sequelize. Example knexfile.js in backend/ root:

```javascript
module.exports = {
  development: {
    client: 'pg',
    connection: process.env.DATABASE_URL,
    migrations: {
      directory: './src/db/migrations'
    },
    seeds: {
      directory: './src/db/seeds'
    }
  },
  production: {
    client: 'pg',
    connection: process.env.DATABASE_URL,
    migrations: {
      directory: './src/db/migrations'
    },
    seeds: {
      directory: './src/db/seeds'
    }
  }
};
```

Migrations Example (backend/src/db/migrations/20230101_init.js):

```javascript
exports.up = function(knex) {
  return knex.schema.createTable('user_sessions', function(table) {
    table.increments('id').primary();
    table.string('session_id').notNullable().unique();
    table.jsonb('selected_concept');
    table.timestamp('created_at').defaultTo(knex.fn.now());
    table.timestamp('updated_at').defaultTo(knex.fn.now());
  });
};

exports.down = function(knex) {
  return knex.schema.dropTable('user_sessions');
};
```

Run migrations:

```
cd backend
npx knex migrate:latest
```

Improved Logging & Error Handling

Use a logging library like winston. In backend/src/utils/logger.js:

```
const { createLogger, transports, format } = require('winston');
const { logLevel } = require('../config/config');

const logger = createLogger({
  level: logLevel,
  format: format.combine(
    format.timestamp(),
    format.json()
  ),
  transports: [
    new transports.Console()
  ]
});

module.exports = logger;
```

Error Handling Middleware (backend/src/middlewares/errorHandler.js):

```
const logger = require('../utils/logger');

module.exports = (err, req, res, next) => {
  logger.error(`Error: ${err.message}`, { stack: err.stack });
  res.status(err.statusCode || 500).json({ error: err.message });
};
```

Use the error handler in server.js:

```
const errorHandler = require('./middlewares/errorHandler');
app.use(errorHandler);
```

Validation & Input Sanitization

In backend/src/utils/validators.js, you might integrate Joi or Zod for schema validation:

```
const Joi = require('joi');

const selectConceptSchema = Joi.object({
  concept: Joi.object({
    title: Joi.string().required(),
    description: Joi.string().required()
  }).required()
});
```

```
module.exports = { selectConceptSchema };
```

Use in routes:

```
const { selectConceptSchema } = require('../utils/validators');

app.post('/api/select-concept', async (req, res, next) => {
  const { error } = selectConceptSchema.validate(req.body);
  if (error) return next(error);

  // Proceed if valid
  const concept = req.body.concept;
  // Save to DB
  const { sessionID } = req;
  await db('user_sessions').update({ selected_concept: concept }).where({
session_id: sessionID });
  res.json({status: 'ok'});
});
```

Production Build & Deployment

Use Docker and docker-compose for a consistent environment:

docker-compose.yml:

```
version: '3.9'
services:
  db:
    image: postgres:13
    environment:
      POSTGRES_USER: user
      POSTGRES_PASSWORD: password
      POSTGRES_DB: mydatabase
    volumes:
      - db_data:/var/lib/postgresql/data
    ports:
      - "5432:5432"

  backend:
    build:
      context: ./backend
      dockerfile: ../docker/Dockerfile.backend
    environment:
      DATABASE_URL: postgres://user:password@db:5432/mydatabase
      CULTURAL_CONTEXT_URL: http://cultural_context_agent:8000/cultural_context
      SESSION_SECRET: super_secret_session_key
    depends_on:
      - db
      - cultural_context_agent
    ports:
      - "3001:3001"

  cultural_context_agent:
    build:
      context: ./ai-services/cultural_context_agent
      dockerfile: Dockerfile
    ports:
      - "8000:8000"

  frontend:
    build:
      context: ./frontend
      dockerfile: ../docker/Dockerfile.frontend
    ports:
      - "3000:3000"
    depends_on:
      - backend

volumes:
  db_data:
```

docker/Dockerfile.backend:

```
FROM node:18-alpine
WORKDIR /usr/src/app
COPY package*.json ./
RUN npm install
COPY . .
ENV NODE_ENV=production
RUN npm run build
EXPOSE 3001
CMD ["npm", "start"]
```

docker/Dockerfile.frontend:

```
FROM node:18-alpine
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
RUN npm run build
EXPOSE 3000
CMD ["npm", "run", "start"]
```

AI Service Dockerfile (e.g. ai-services/cultural_context_agent/Dockerfile):

```
FROM python:3.10-slim
WORKDIR /app
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt
COPY . .
EXPOSE 8000
CMD ["uvicorn", "app:app", "--host", "0.0.0.0", "--port", "8000"]
```

Security & Authentication

For a public-facing festival site, consider adding:

- CSRF protection in forms (e.g., csurf middleware in Express).
- Rate limiting (e.g., express-rate-limit) to prevent abuse of AI endpoints.
- HTTPS termination (via a reverse proxy like Nginx or a load balancer).
- Basic user accounts if needed (allowing local storytellers to log in and save work with a proper login system).

Example adding rate limiting in backend/server.js:

```
const rateLimit = require('express-rate-limit');

const limiter = rateLimit({
  windowMs: 10 * 60 * 1000, // 10 minutes
  max: 100,
});

app.use('/api/', limiter);
```

Linting & Code Quality

Use ESLint and Prettier in the Node.js and React code:

```
npm install --save-dev eslint prettier
```

Configure .eslintrc.js and run npm run lint before committing. Use a pre-commit hook (e.g., with Husky) to ensure code standards are met.

CI/CD Integration

In a production scenario, use a CI pipeline (GitHub Actions, GitLab CI, or CircleCI) to:

- Run tests (unit and integration tests)
- Run linting and build steps
- Run container builds
- Deploy to staging, then production

Example GitHub Action .github/workflows/ci.yml:

```
name: CI
on:
  push:
    branches: [ "main" ]
jobs:
  build-test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
      - name: Setup Node
        uses: actions/setup-node@v2
        with:
          node-version: '18'
      - run: cd backend && npm install && npm run lint && npm test
      - run: cd frontend && npm install && npm run lint && npm run build
```

Summary of Production-Readiness Improvements

- **Environment Variables & Config**: Managing secrets and URLs outside code.
- **Database Integration & Migrations**: Persistent storage of user session data and story progress.
- **Logging & Error Handling**: Structured logs and a global error handler.

- **Input Validation**: Using schemas to validate incoming data.
- **Dockerization & CI/CD**: Containerized services and automated testing and deployment.
- **Security Hardening**: Rate limiting, possibly authentication and HTTPS.
- **Linting & Code Style**: Ensuring code cleanliness and maintainability.

By implementing these steps and best practices, the project is closer to a stable, maintainable, and scalable production-ready codebase. It sets the foundation for a reliable, user-friendly platform that can support community engagement at the Quandamooka Film Festival and beyond.

Part 6: Final Code Report, Towards A Turnkey Product

Below is a more holistic, "turnkey" example that brings together the previously discussed concepts into a production-level baseline. While it's still not a fully complete and tested codebase (due to the complexity and time constraints here), it is a strong template from which a real production system could be built. We'll present a cohesive file and directory structure, key code files, Docker configurations, and deployment instructions that—if fleshed out and properly configured—would allow a development team to launch a stable, production-grade system.

This leverages:

- **Infrastructure as Code** via Docker and docker-compose.
- **Secure Config** management via environment variables.
- **Logging, Error Handling, Validation, and Rate Limiting**.
- **Database migrations & seeds** for persistent data.
- **Automated test scaffolding** for CI/CD pipelines.
- **Scalable microservices architecture**: A Node.js backend orchestrating Python-based AI services.
- **Frontend (React)** with accessible UI components.

Final Directory Structure

```
project-root/
  frontend/
    src/
      pages/
        Home.js
        Ideation.js
        Characters.js
        ScriptDrafting.js
        ...
      components/
        KooAssistant.js
        Layout.js
        ...
      styles/
        globals.css
      utils/
        api.js
      ...
    public/
      icons/
      ...
    package.json
    .env (frontend if needed, minimal)
    ...

  backend/
    src/
      config/
        config.js
      db/
        migrations/
          20230101_init.js
        seeds/
        knexfile.js
      middlewares/
        errorHandler.js
        notFoundHandler.js
      routes/
        index.js
        concepts.js
        characters.js
        script.js
        visuals.js
        ...
      services/
        aiClient.js
        dbClient.js
        sessionService.js
      utils/
        logger.js
        validators.js
      tests/
        concepts.test.js
      server.js
    package.json
    .env

  ai-services/
    cultural_context_agent/
      app.py
      requirements.txt
      Dockerfile
    plot_structure_agent/
      app.py
      requirements.txt
      Dockerfile
    scriptwriting_agent/
      app.py
      requirements.txt
      Dockerfile
    ...
  docker/
    Dockerfile.backend
    Dockerfile.frontend
    docker-compose.yml

  docs/
    README.md
    ARCHITECTURE.md
    DEPLOYMENT.md

  .github/workflows/ci.yml
  .env (root if needed only for local dev)
```

Key Backend Code Files

backend/src/config/config.js
Central place for env configs:

```
require('dotenv').config();

module.exports = {
  port: process.env.PORT || 3001,
  databaseUrl: process.env.DATABASE_URL,
  aiServices: {
    culturalContextUrl: process.env.CULTURAL_CONTEXT_URL ||
'http://cultural_context_agent:8000/cultural_context',
    plotStructureUrl: process.env.PLOT_STRUCTURE_URL ||
'http://plot_structure_agent:8001/plot_structure',
    scriptWriterUrl: process.env.SCRIPT_WRITER_URL ||
'http://scriptwriting_agent:8002/script',
    // Add more services as needed
  },
  sessionSecret: process.env.SESSION_SECRET || 'super_secret_session_key',
  logLevel: process.env.LOG_LEVEL || 'info',
  nodeEnv: process.env.NODE_ENV || 'development',
};
```

backend/src/server.js
Main entry point of the Node backend:

```
const express = require('express');
const session = require('express-session');
const cors = require('cors');
const helmet = require('helmet');
const rateLimit = require('express-rate-limit');
const { port, sessionSecret, nodeEnv } = require('./config/config');
const router = require('./routes/index');
const errorHandler = require('./middlewares/errorHandler');
const notFoundHandler = require('./middlewares/notFoundHandler');
const logger = require('./utils/logger');
const KnexSessionStore = require('connect-session-knex')(session);
const knex = require('knex')(require('./db/knexfile')[nodeEnv]);

const app = express();
app.use(express.json());
app.use(cors());
app.use(helmet());

const limiter = rateLimit({
  windowMs: 10 * 60 * 1000,
  max: 100,
});
app.use('/api/', limiter);

const store = new KnexSessionStore({
  knex,
  tablename: 'sessions'
});

app.use(session({
  secret: sessionSecret,
  resave: false,
  saveUninitialized: true,
  store,
  cookie: { secure: nodeEnv === 'production', maxAge: 86400000 }
}));

app.use('/api', router);

// 404 handler
app.use(notFoundHandler);

// Error handler
app.use(errorHandler);

app.listen(port, () => {
  logger.info(`Backend server running on port ${port}`);
});
```

backend/src/routes/index.js
Collects all route files:

```
const express = require('express');
const router = express.Router();
const conceptsRouter = require('./concepts');
// Add other routers like characters, script, etc.

router.use('/concepts', conceptsRouter);
// router.use('/characters', charactersRouter);
// router.use('/script', scriptRouter);
// ...

module.exports = router;
```

backend/src/routes/concepts.js
A sample endpoint using AI service:

```
const express = require('express');
const router = express.Router();
const axios = require('axios');
const { aiServices } = require('../config/config');
const { selectConceptSchema } = require('../utils/validators');
const logger = require('../utils/logger');
const db = require('../services/dbClient');

router.get('/', async (req, res, next) => {
  try {
    const response = await axios.post(aiServices.culturalContextUrl, { locale: 
'Minjerribah', theme: 'community' });
    res.json(response.data.concepts);
  } catch (err) {
    next(err);
  }
});

router.post('/select', async (req, res, next) => {
  try {
    const { error } = selectConceptSchema.validate(req.body);
    if (error) return next(error);

    const concept = req.body.concept;
    await db('user_sessions')
      .insert({ session_id: req.sessionID, selected_concept: concept })
      .onConflict('session_id').merge();

    res.json({status: 'ok'});
  } catch (err) {
    next(err);
  }
});

module.exports = router;
```

backend/src/db/knexfile.js
Knex config:

```
require('dotenv').config();
module.exports = {
  development: {
    client: 'pg',
    connection: process.env.DATABASE_URL,
    migrations: {
      directory: './src/db/migrations'
    },
    seeds: {
      directory: './src/db/seeds'
    }
  },
  production: {
    client: 'pg',
    connection: process.env.DATABASE_URL,
    migrations: {
      directory: './src/db/migrations'
    },
    seeds: {
      directory: './src/db/seeds'
    }
  }
};
```

backend/src/db/migrations/20230101_init.js
Example migration:

```
exports.up = function(knex) {
  return knex.schema.createTable('user_sessions', table => {
    table.increments('id').primary();
    table.string('session_id').notNullable().unique();
    table.jsonb('selected_concept');
    table.timestamps(true, true);
  }).createTable('sessions', table => {
    // Used by connect-session-knex
    table.string('sid').primary();
    table.json('sess').notNullable();
    table.timestamp('expired').notNullable();
  });
};

exports.down = function(knex) {
  return knex.schema.dropTableIfExists('user_sessions')
    .dropTableIfExists('sessions');
};
```

AI Services Example (Cultural Context Agent)

ai-services/cultural_context_agent/app.py:

```
from fastapi import FastAPI
from pydantic import BaseModel
import uvicorn

class InputData(BaseModel):
    locale: str
    theme: str

app = FastAPI()

@app.post("/cultural_context")
async def cultural_context(data: InputData):
    # In a production scenario, call LLM. Here we mock:
    concepts = [
        {"title": "Island Sunrise Story", "description": "A family welcomes a new
day with local traditions."},
        {"title": "Festival of Voices", "description": "Community members gather
for stories and music."},
        {"title": "Tides of Change", "description": "As visitors arrive, cultures
blend and evolve."}
    ]
    return {"concepts": concepts}

if __name__ == "__main__":
    uvicorn.run(app, host="0.0.0.0", port=8000)
```

Frontend Integration

frontend/src/utils/api.js:
A simple API helper:

```
import axios from 'axios';

const api = axios.create({
  baseURL: process.env.REACT_APP_API_URL || 'http://localhost:3001/api',
});

export default api;
```

frontend/src/pages/Home.js:
Uses simple links and routing:

```
import React from 'react';
import { Link } from 'react-router-dom';

export default function Home() {
  return (
    <div className="p-10 flex flex-col items-center">
      <h1 className="text-4xl mb-6 font-bold">Create Your Story!</h1>
      <div className="flex flex-col gap-4">
        <Link to="/ideation" className="bg-green-500 text-white px-6 py-3 rounded-
lg text-2xl">Start New Story</Link>
        <Link to="/resume" className="bg-yellow-500 text-white px-6 py-3 rounded-lg
text-2xl">Continue Where I Left Off</Link>
        <Link to="/gallery" className="bg-blue-500 text-white px-6 py-3 rounded-lg
text-2xl">Explore Finished Stories</Link>
      </div>
    </div>
  );
}
```

frontend/src/pages/Ideation.js:

```
import React, { useEffect, useState } from 'react';
import api from '../utils/api';
import { useHistory } from 'react-router-dom';

export default function Ideation() {
  const [concepts, setConcepts] = useState([]);
  const [selected, setSelected] = useState(null);
  const history = useHistory();

  useEffect(() => {
    api.get('/concepts')
      .then(res => setConcepts(res.data))
      .catch(err => console.error(err));
  }, []);

  const handleSelect = (concept) => {
    api.post('/concepts/select', { concept })
      .then(() => {
        setSelected(concept);
      })
      .catch(err => console.error(err));
  };

  const handleNext = () => {
    if (selected) history.push('/characters');
  };

  return (
    <div className="p-8">
      <h2 className="text-3xl font-semibold mb-4">What's Your Story About?</h2>
      <div className="flex space-x-4 overflow-x-scroll">
        {concepts.map((c, i) => (
          <div key={i} className="min-w-[200px] bg-white shadow p-4 rounded">
            <h3 className="text-xl font-bold">{c.title}</h3>
            <p>{c.description}</p>
            <button onClick={() => handleSelect(c)} className="mt-2 bg-green-500
px-4 py-2 text-white rounded">Use This Idea!</button>
          </div>
        ))}
      </div>
      {selected && (
        <div className="mt-6">
          <p className="text-xl">You selected: {selected.title}</p>
          <button onClick={handleNext} className="bg-blue-500 text-white px-4 py-2
rounded mt-4">Next: Characters</button>
        </div>
      )}
    </div>
  );
}
```

Docker & Deployment

docker/docker-compose.yml:

```
version: '3.9'
services:
  db:
    image: postgres:13
    environment:
      POSTGRES_USER: user
      POSTGRES_PASSWORD: password
      POSTGRES_DB: mydatabase
    volumes:
      - db_data:/var/lib/postgresql/data
    ports:
      - "5432:5432"

  cultural_context_agent:
    build:
      context: ../ai-services/cultural_context_agent
      dockerfile: Dockerfile
    ports:
      - "8000:8000"

  plot_structure_agent:
    build:
      context: ../ai-services/plot_structure_agent
      dockerfile: Dockerfile
    ports:
      - "8001:8001"

  scriptwriting_agent:
    build:
      context: ../ai-services/scriptwriting_agent
      dockerfile: Dockerfile
    ports:
      - "8002:8002"

  backend:
    build:
      context: ../backend
      dockerfile: ../docker/Dockerfile.backend
    environment:
      DATABASE_URL: postgres://user:password@db:5432/mydatabase
      CULTURAL_CONTEXT_URL: http://cultural_context_agent:8000/cultural_context
      PLOT_STRUCTURE_URL: http://plot_structure_agent:8001/plot_structure
      SCRIPT_WRITER_URL: http://scriptwriting_agent:8002/script
      SESSION_SECRET: super_secret_session_key
    depends_on:
      - db
      - cultural_context_agent
      - plot_structure_agent
      - scriptwriting_agent
    ports:
      - "3001:3001"
    command: [ "npm", "run", "start:prod" ]

  frontend:
    build:
      context: ../frontend
      dockerfile: ../docker/Dockerfile.frontend
    environment:
      REACT_APP_API_URL: http://localhost:3001/api
    depends_on:
      - backend
    ports:
      - "3000:3000"

volumes:
  db_data:
```

docker/Dockerfile.backend:

```
FROM node:18-alpine
WORKDIR /usr/src/app
COPY package*.json ./
RUN npm install --only=production
COPY . .
RUN npx knex migrate:latest
EXPOSE 3001
CMD ["npm", "run", "start:prod"]
```

docker/Dockerfile.frontend:

```
FROM node:18-alpine
WORKDIR /app
COPY package*.json ./
RUN npm install --only=production
COPY . .
RUN npm run build
# Serve with a lightweight server like serve
RUN npm install -g serve
EXPOSE 3000
CMD ["serve", "-s", "build"]
```

Deployment:

- Ensure .env files or environment variables are correctly set in your CI/CD or hosting environment.
- Run: docker-compose up --build to start the entire stack locally.
- Access the frontend via http://localhost:3000, backend via http://localhost:3001/api.
- Scale services as needed using docker-compose scale frontend=3.

Additional Production Considerations

- **Load Balancing & Reverse Proxy**: Use NGINX or a cloud load balancer in front of the frontend and backend for SSL termination and routing.
- **Monitoring & Metrics**: Integrate Prometheus/Grafana or a cloud service for tracking performance and health.
- **Continuous Integration/Deployment**: The .github/workflows/ci.yml pipeline can run tests, lint, build Docker images, and push to a registry. Then a CD pipeline can deploy updated containers to a production cluster.
- **Secrets Management**: Store sensitive info (DATABASE_URL, SESSION_SECRET) in a secret manager like AWS Secrets Manager or HashiCorp Vault, not in .env.
- **High Availability**: Deploy the containers on Kubernetes or ECS, use autoscaling, and attach persistent storage volumes for the database.

Outcome:
This approach, code structure, and configuration provide a turnkey, production-level starting point. By following these patterns—modular code, strong validation, environment-driven config, logging, containerization, and CI/CD integration—you have a reliable foundation to build upon. From here, a real-world team would fill in any gaps, refine the UI, add rigorous tests, and integrate robust AI model prompts to deliver a polished, stable experience for the Quandamooka Film Festival community and beyond.