ASHESI
UNIVERSITY

DIGITAL ILLUSTRATION TOOL FOR COMICS BOOK CREATION AND COLLABORATION

Applied Capstone

B.Sc. Computer Science

Nyameye Akumia

2025

ASHESI UNIVERSITY

DIGITAL ILLUSTRATION TOOL FOR COMICS BOOK CREATION AND COLLABORATION

APPLIED CAPSTONE

Applied Capstone submitted to the Department of Computer Science & Information Systems, Ashesi University in partial fulfilment of the requirements for the award of Bachelor of Science degree in Computer Science.

Nyameye Akumia

2025

DECLARATION

I hereby declare that this Applied Capstone is the result of my own original work and that no part of it has been presented for another degree in this university or elsewhere.

Candidate's Signature: 

.....

Candidate's Name:

Nyameye Akumia

.....

Date:

28/04/2025

.....

I hereby declare that preparation and presentation of this Applied Capstone was supervised in accordance with the guidelines on supervision of Applied Capstone laid down by Ashesi University.

Supervisor's Signature: 

.....

Supervisor's Name: Eyram Tawia

.....

Date: 28/04/2025

.....

Acknowledgements

I would like to express my sincere gratitude to all those who contributed to the successful completion of this capstone project.

First, I extend my appreciation to Mr. Eyram Tawia, my capstone supervisor, for his invaluable guidance, feedback, and encouragement throughout the development process. His insights and support were instrumental in shaping both the technical and conceptual direction of the project.

I am also grateful to Ashesi University for providing the academic environment, resources, and support that made this project possible. The knowledge, skills, and values cultivated during my time at Ashesi have been essential in bringing this work to fruition.

Finally, I would like to acknowledge the support of my friends and family, whose encouragement and understanding played an important role in helping me stay focused and motivated throughout this project.

To everyone who contributed to this journey, your support is deeply appreciated.

Abstract

The growing adoption of digital technologies across creative industries has revolutionized collaboration, yet comic book production workflows have remained largely fragmented and inefficient. Traditional methods rely on a patchwork of disconnected tools for writing, illustration and project management. This capstone project addresses these challenges by developing the Digital Comics Illustration Tool, a cloud-hosted, real-time collaboration platform specifically designed for end-to-end comic creation.

The system architecture follows a modular client-server model optimized for scalability, real-time responsiveness, and maintainability. The frontend is developed using React, offering an intuitive, role-specific interface that adapts to writers, artists, and editors. The backend, built with Node.js and Express, coordinates data flow between client interfaces, a Firestore NoSQL database for real-time data synchronization, and a MySQL database for structured storage. The project demonstrates the feasibility of centralizing comic production workflows within a cloud-native platform and highlights the benefits of tightly integrated, collaborative tools in improving creative efficiency, consistency, and project quality.

Keywords: Digital Comics, Real-Time Collaboration, Cloud Computing, Human-Computer Interaction (HCI), Creative Arts Technology, Comic Production Tools, AI-Assisted Design, Google Cloud Platform, React Applications, Node.js Backend

Table of Contents

List of Figures

Chapter 1: Introduction

The creative industries are undergoing a profound digital transformation that is reshaping how artists work together. Advances in cloud computing and networked applications have enabled creators to collaborate beyond the confines of physical studios, sparking new modes of co-creation . Digital-first workflows now allow distributed teams to co-author content in real time, blurring geographic boundaries. In fields like graphic design and writing, cloud-based collaborative tools (e.g., Google Docs for text, Figma for design) have become commonplace, offering shared workspaces and version histories to multiple users simultaneously ([3]). These tools exemplify the modern paradigm where concurrency and iteration are seamlessly supported – multiple contributors can edit, comment, and revert changes on a shared artifact with minimal friction ([11, 8]).

1.1 Background

Comic book production, a traditionally sequential art form involving writers, pencillers, inkers, colorists, and letterers, is beginning to experience this digital shift. Webcomics and digital comic platforms illustrate the convergence of participatory culture with new technology, enabling global authors and artists to work together in novel ways. For instance, the rise of webtoon platforms has shown how creators from different regions can collaboratively produce content directly for a worldwide audience, transcending established print workflows ([2]). However, despite these signs of change, the comic creation process has not yet fully capitalized on collaborative software innovations to the extent seen in other creative fields. The multi-modal nature of comics – combining narrative text and visual art – poses unique challenges for co-creation, and existing tools often remain siloed for individual tasks (writing, illustrating, editing) rather than integrated for end-to-end teamwork.

1.2 Problem Definition

Fragmented Workflows. Comic production typically involves a chain of specialized steps often carried out with disparate software tools. In practice, creators juggle word processors for scripts, digital drawing programs for artwork, and separate layout or publishing software to compile pages ([1]). The lack of a unified platform means creative teams have to manually synchronize changes. Industry surveys highlight that chasing down inputs and approvals across different channels is a major slowdown in creative processes . The need to piece together multiple applications (each handling only one part of the workflow) underscores a gap in workflow integration for sequential art projects.

Versioning Issues. Without robust version control, creative teams often face confusion over which draft of a script or which iteration of an artwork is the "current" version. Traditional software versioning systems (like Git) are ill-suited to non-code assets such as images and rich text, so artists resort to ad-hoc practices like saving sequential files (for example, "page_v1.png", "page_final_final.png") or relying on cloud storage history. Recent HCI research confirms that while version control features are increasingly present in modern creative tools (e.g., timestamped history in Adobe or cloud backups), they are still geared toward linear progress and efficiency rather than the exploratory, nonlinear process of art-making ([6]). Sterman et al. observe that creative practitioners repurpose versioning features in unexpected ways – treating saved states as a palette of alternatives, and using forks of a design to explore variations – uses that typical version control interfaces don't fully support ([9]).

Remote Collaboration. Challenges Collaboration in comics has historically been asynchronous – writers and artists often work separately, sending scripts and artwork back and forth via email or cloud drives. As teams become increasingly remote and globally distributed, this asynchronous, file-based exchange struggles to keep up with the speed and interactivity desired. A recent industry report noted that 75% of creative collaboration in marketing and design now occurs remotely, yet many creatives feel that "collaboration is harder when working remotely," citing the loss of informal in-person communication ([10,

5]). There is a documented need for collaboration platforms that provide a shared visual workspace.

The problem this capstone addresses is: **How can we design a cloud-based platform that integrates AI to streamline the comic creation workflow, provides robust version management, and enables real-time remote collaboration for all roles in a comic production team?**

1.3 Project Objectives

The overarching aim of this project is to build a *cloud-based, AI-assisted, real-time collaborative comic creation platform* that directly tackles the aforementioned problems. The specific objectives are:

- **AI-Assisted Script-to-Panel Parsing** – Develop an AI module that can parse a comic script (text narrative or screenplay-like input) and intelligently suggest panel breakdowns, layouts, and initial scene compositions. This involves natural language understanding to identify scene descriptions, characters, and dialogue, and computer vision techniques to generate storyboard-like representations.

- **Real-time Shared Canvas for Collaborative Illustration** – Implement a cloud synchronized drawing canvas where multiple artists (penciller, inker, colorist, etc.) can work on the same page simultaneously from different locations. This includes developing concurrency control so that collaborators can see each other's strokes and edits in real time, with locking or partitioning as needed (for example, one artist coloring while another inks, on separate layers). The shared canvas should support layered artwork, annotations, and live chat or voice integration to mimic the experience of a joint studio session. This real-time co-editing environment aims to eliminate the lag of turn-based collaboration and enable instantaneous feedback and co-creation.

- **Integrated Version Control and History Management** – Provide built-in version control tailored to comics. Every change (script edits, artwork revisions, color adjustments) should be recordable as a version state, with the ability to branch (e.g., try

an alternative art style on a duplicate of a page), merge contributions, and roll back to earlier iterations. The interface will present version history in an intuitive visual manner (for instance, a timeline with thumbnails of page states) to help creative users navigate revisions. This objective draws on the concept of Creative Version Control Systems , aiming to give comic teams both the safety net of non-destructive editing and a sandbox for creative exploration. Contributors will be able to comment on or compare different versions, streamlining the review and approval cycles.

Together, these objectives form a comprehensive approach to building a platform that not only merges currently fragmented tools into one ecosystem but also introduces intelligent assistance and real-time co-working capabilities.

1.4 Significance of the Study

Sequential art holds a unique place in the creative spectrum, blending literary and visual art into a cohesive narrative medium. Despite its popularity and the proliferation of digital drawing tools, there is a conspicuous gap in specialized collaboration solutions for this medium. Most comic creators today either work individually or adopt improvised workflows to collaborate. Mainstream art software has only recently begun to acknowledge collaborative use cases: for example, Clip Studio Paint (a leading digital comic art tool) introduced a "Teamwork" cloud feature to let multiple people work on a comic project together online ([12]), but this functionality is still rudimentary, preventing simultaneous editing on the same page and requiring all collaborators to have high-end licenses. In essence, the current tools were not originally designed for synchronous co-creation of comics, and their retrofitted collaboration features remain limited.

This project's significance lies in addressing an unmet need in the intersection of HCI and creative arts: a dedicated platform for collaborative comic creation. By bringing real-time collaboration and AI assistance into the comic workflow, the project stands to significantly enhance the efficiency and creative quality of producing sequential art. Key anticipated benefits include:

- Streamlined Production Pipeline

- Faster Iteration and Enhanced Creativity
- Improved Remote Collaboration Experience
- Filling a Research and Industry Gap

1.5 Literature Review

To ground the project in existing knowledge, this section reviews relevant literature across four pertinent areas: (a) HCI studies on digital creative workflows, (b) collaborative systems and co-authoring tools, (c) applications of AI in visual storytelling and digital comics, and (d) existing comic creation tools and their limitations.

1.5.1 HCI in Digital Creative Workflows

Human-Computer Interaction (HCI) research has increasingly turned attention to how digital tools can support creative work. As creative professionals adopt software for activities like graphic design, writing, and music composition, researchers have studied the evolving workflow practices and pain points. A recurring theme is the management of intermediate creative stages and revisions. Sterman et al. (2022) conducted an in-depth study on how creative practitioners use version histories in their work, highlighting that traditional version control concepts (originating from software engineering) don't perfectly align with creative processes ([5]). They found that artists and designers use versioning not just to "roll back" errors, but as a creative tool—saving off variations, combining elements from different iterations, and even repurposing old ideas later in a project . This suggests that tools need to be more flexible in how they capture and present the evolution of creative artifacts. The notion of Creative Version Control Systems (CVCS) emerged from this discourse: systems that would allow non-linear version trees, visual diffing of images or multimedia, and context-rich history to support the way creators actually work ([2]).

HCI research has also examined general collaborative creativity. Studies in Computer-Supported Cooperative Work (CSCW) suggest that providing shared representations (such as a common whiteboard or canvas) is vital for grounding collaboration among team mem-

bers, especially when they are remote ([2]). Mutual awareness of who is doing what can prevent conflicts and enhance a feeling of working together. This justifies features in our platform like live cursors or highlights showing each collaborator's focus area on a page. Additionally, maintaining a chat log or comments linked to specific panels can serve as the persistent record of design rationale – a concept borrowed from design rationale management tools in HCI . In essence, the interdisciplinary insights from HCI underscore that the platform must carefully design its interaction patterns to support the creative flow (flexible versioning, AI assistance that preserves human agency, and rich real-time communication channels).

1.5.2 Collaborative Systems and Co-Authoring Tools

Collaborative software systems have been extensively studied and developed for text editing, software coding, and general teamwork (e.g., project management). In the context of co-authoring creative content, notable systems include real-time collaborative editors like Google Docs (for text) and Overleaf (for academic writing), as well as design collaboration platforms like Figma and Miro. These systems provide valuable lessons applicable to a comic creation platform. First, they demonstrate the importance of immediate consistency – all users seeing updates in near real-time – to keep collaborators on the same page and sustain momentum. Our platform's architecture will leverage web real-time frameworks to achieve this consistency on the shared canvas. Second, they incorporate conflict resolution strategies. Google Docs, for instance, allows simultaneous edits by different users and merges changes character-by-character, whereas design tools like Figma lock an object when one user is manipulating it to avoid clashing edits. For comics, a hybrid approach may be needed: two colorists can't paint the exact same area at once without conflict, so a form of floor control (temporary locks or reservations on a panel or layer) might be implemented, echoing the locking mechanismintroduced in Clip Studio's Teamwork feature to prevent concurrent edits on the same page region ([2]).

Version control in co-authoring is another critical aspect. Systems like wikis track every edit for accountability and allow reverts, and collaborative coding platforms (GitHub)

use branching/merging to manage parallel contributions. The literature suggests that non-technical users can be overwhelmed by complex branching models; thus a visual approach (timeline sliders, forks represented graphically) as explored in some research prototypes would be beneficial . A 2023 industry white paper by Adobe also emphasizes integrations across tools – for example, Creative Cloud now integrates with Microsoft Teams/Slack for easier sharing – highlighting that collaboration tools should fit into users' broader ecosystem. Therefore, our platform might consider export/import capabilities or API hooks to common services (e.g., exporting a completed comic to a PDF publisher, or integrating with project management boards).

Co-authoring in storytelling has some precedent in research: Mencarini et al. (2015) evaluated a tablet-based collaborative comic composition app for co-located users and found that providing a constrained library of assets and sentences enabled two authors speaking different languages to create a comic together effectively . While their scenario (multilingual storytelling with fixed assets) differs from our professional production focus, it underlines the value of a shared interface and guidance to facilitate collaboration. Similarly, in writing, studies on collaborative creative writing tools (e.g., co-writing fiction with shared editors) note the need for role negotiation and awareness – one person might brainstorm while another edits language, akin to writer–editor roles in comics. Role-based access control models from CSCW literature can inform how we allow certain actions for certain roles (for instance, locking the script from direct edits by an artist during a later production stage, to enforce a pipeline discipline if desired).

Finally, the surge in remote collaboration due to the pandemic has been captured in industry surveys. A Filestage report (2023) on creative collaboration found that teams consider "**too many tools**" and context switching as a key challenge, and that simplifying the number of platforms significantly improves collaboration speed . This supports the core concept of our project: by consolidating multiple functions into one platform, we reduce the cognitive and operational overhead for the team. The same report also noted that fully in-office teams did not necessarily collaborate faster than remote/hybrid teams, debunking a myth and suggesting that with the right tools, remote collaboration can be equally if not

more efficient . It is an encouraging data point for building robust remote collaboration capabilities – the tool can truly make a difference.

1.5.3 Existing Tools for Comic Creation and Their Limitations

A scan of existing software tools for comic creation reveals a spectrum ranging from professional-grade art programs to niche web applications. On the professional end, tools like **Clip Studio Paint, Adobe Photoshop/Illustrator**, and Procreate are popular for drawing and page layout. They offer powerful drawing capabilities and, in the case of Clip Studio, specialized features like comic panel rulers, speech bubble tools, and multi-page management. However, historically these have been single-user applications. Collaboration, if any, was implemented via external file sharing. As noted, Clip Studio Paint's recent Teamwork update (2021) allows multiple contributors to share a project via the cloud , but it essentially serializes collaboration (only one person can edit a given page at a time, others are locked out) rather than enabling true simultaneous co-editing. Adobe has introduced cloud documents and invited editing in Photoshop, yet this is still in early stages and often focuses on design review rather than real-time joint drawing.

There are also web-based comic makers oriented toward beginners or educators, such as Pixton, Storyboarder, and various mobile apps. These typically provide drag-and-drop interfaces with premade assets (characters, backgrounds) to compose simple comic strips. While useful for quick storyboarding or non-artists, they are not suitable for professional creators who want full control over the artwork style and story complexity. Moreover, such tools rarely support multi-user collaboration; they are single-user by design, meant to simplify creation for one user at a time. A literature example is Comic Spin (referenced earlier), which was designed for a specific use case (helping people with aphasia create comics) and used constrained text and imagery to guide the user . It was not intended for multiple collaborators or high-fidelity comic production, which highlights the gap for professional tools.

Another category to consider is project management and version control solutions that some comic teams informally use. For instance, teams might use Dropbox or Google

Drive to share files, Trello or other Kanban boards to track progress (”page 1 script done”, ”page 1 inks WIP”, etc.), and Git or SVN if they are technically inclined to version control their files. These tools are generic and not tailored to comics, leading to friction. The literature on creative collaboration emphasizes integration – when the tools are not integrated, users act as the “glue” between them, which can cause errors or extra work . For example, a comic editor might manually ensure the artist is using the latest script file from Drive, or maintain an external spreadsheet to track which version of an image was sent to the letterer. Such overhead would be reduced by an integrated platform that has a single source of truth and built-in project tracking visible to all team members.

Chapter 2: Requirements Analysis and Specification

This chapter outlines the requirements for the Digital Comics Illustration Tool, based on the identified problem, project objectives, and industry-specific workflows. It begins by identifying the key stakeholders involved in the comic creation pipeline, followed by a detailed use case analysis, system requirements, and the proposed implementation structure. As this capstone is an applied project and not human-subject research, no ethical clearance or IRB application was necessary. However, relevant ethical concerns related to data handling and collaboration integrity are discussed in the final section.

2.1 Key Stakeholders

The cloud-based digital comics creation tool serves a range of stakeholders involved in the creation and management of digital comic content. The primary stakeholders include the **comic creators**, such as illustrators and writers, who use the platform to collaboratively produce comic panels and narratives. These creators rely on the tool's features for real-time editing, asset management, and version control to streamline their workflow. Another important group is **editorial and production staff**, including editors or art directors who oversee quality and continuity. They use the system to review changes, provide feedback, and ensure the final comic meets publishing standards. In a collaborative project setting, **team collaborators** (co-writers, co-artists, letterers, etc.) are key stakeholders as well – the tool must support their simultaneous contributions and communication needs. Additionally, **project clients or publishers** can be considered stakeholders when the comics are created for commercial release; they are concerned with the timely delivery, security, and quality of the creative content. Finally, the **platform administrators or developers** (considered in the project context) are stakeholders responsible for maintaining the system, ensuring it meets user needs and ethical standards. Each of these stakeholders has distinct requirements and interests that inform the design and features of the proposed system ([2, 11, 12]).

2.2 System Requirements

Based on the stakeholder needs and use cases above, the system is defined by a set of functional and non-functional requirements. The functional requirements describe what the system should do, i.e., the features and capabilities necessary for the comics creation tool. The non-functional requirements outline the expected qualities and constraints of the system (performance, security, usability, etc.) that ensure it operates effectively in a cloud environment.

2.2.1 Functional Requirements

- **Real-Time Collaboration:** The system shall support multiple users editing the same comic project simultaneously. Updates made by one user (e.g., drawing a line or adding text) should be propagated in real time to all other online collaborators viewing that project.

- **Concurrent Editing Conflict Resolution:** The system shall include mechanisms to handle concurrent edits without corrupting the comic data. For example, if two users edit the same panel or text bubble, the system should merge changes in a sensible way or prompt users to resolve conflicts (possibly by locking certain elements while being edited).

- **Version History Management:** The system shall maintain a version history for each comic project. Users can revert to or reference any prior version. Each version entry should record metadata such as timestamp and author of changes, and optionally allow a description of the changes.

- **Drawing and Design Tools:** The platform shall provide a rich set of comic creation tools. This includes freehand drawing on a digital canvas (with support for pen, brush, shapes, eraser, etc.), text tool for adding/editing dialogue or captions and layout tools for arranging panels on a page.

- **User Authentication and Project Access Control:** The system shall require users to

register and login securely. It should implement access controls for projects. It should implement access controls for projects. Only authorized users can access a given comic project. This also includes permission levels; for instance, some collaborators might have view/comment rights but not edit rights as needed.

- **Project Export/Import:** Users shall be able to export the comic in common formats. For example, export an entire comic issue as a PDF or as a sequence of images such that the issue can be printed and shared easily.
- **Responsive Web Interface**

2.2.2 Non-Functional Requirements

- **Usability:** The tool should have an intuitive and user-friendly interface. Comic creators, who may not be tech experts, should find the collaboration and version features easy to use. This implies a gentle learning curve, with clear visual indicators (e.g., showing who is online and where they are editing, highlighting recent changes, etc.). Usability also means providing a smooth drawing experience (minimal lag between input and rendering) to simulate the responsiveness of drawing on paper or local applications.
- **Performance:** Real-time editing operations must be low-latency to maintain the flow of creative work. The system should handle frequent updates (such as continuous brush strokes) efficiently. Additionally, loading a project (especially one with many pages or high-resolution images) should be optimized so that users aren't waiting excessively to start work. Server-side, the architecture must scale to handle multiple projects and collaboration sessions in parallel without degradation in response time.
- **Reliability and Availability:** The cloud service should be highly available and reliable. Users expect that their work will not be lost – thus, the system should implement autosave functionality (periodically saving work-in-progress to the cloud) and backup mechanisms for version history. In case of any network disruption, no data should be lost; the client can cache changes and sync when reconnected. The system should

gracefully handle minor internet connectivity issues and merge offline edits when possible to ensure continuous productivity.

- **Scalability:** As the number of users and projects grows, the system should scale accordingly. This includes scaling storage for possibly thousands of high-resolution images and pages, as well as scaling the real-time collaboration server to support many simultaneous editing sessions. A cloud-native design (using scalable infrastructure or services) would allow dynamic adjustment of resources to meet demand. Scalability also pertains to project size – the system should support comics of substantial length (e.g., a graphic novel with many chapters) without performance loss.

- **Security:** Security is paramount given that creative assets are intellectual property. The system must protect user data both in transit and at rest. This means using secure communication protocols (HTTPS/WSS for data and realtime messages), encryption of stored assets and project files on the server, and robust authentication to prevent unauthorized access. Regular security audits and updates are expected to guard against vulnerabilities. Additionally, user account data and personal information should be safeguarded according to privacy best practices (e.g., hashed passwords, GDPR compliance if relevant).

- **Data Integrity:** The platform should ensure consistency and integrity of the comic data. With concurrent edits and numerous versions, it's important that the underlying data models (for drawings, text, layout) remain consistent (no corruption or inadvertent overwrites). Transactions or locking may be used to maintain integrity when multiple operations happen at once. Every saved version should accurately reflect the state of the project at that time without random anomalies.

- **Maintainability and Extensibility:** The system should be built in a modular way such that it's maintainable by developers and can be extended with new features (like adding new drawing tools or perhaps AI-assisted features in the future). This means clear separation between front-end client code, back-end services, and perhaps a well-documented API that the different components use. Maintainability ensures that bug

fixes and updates can be applied without breaking the system, which is crucial for a long-lived platform.

- **Ethical Compliance:** (Non-functional from a process standpoint) The system's design should comply with ethical guidelines, particularly if AI components are integrated. For example, if an AI module is used for content recommendations or automatic coloring, it should be evaluated for bias or inappropriate suggestions. This ties into the ethical considerations discussed later, but is listed as a requirement to emphasize that the system will incorporate ethical safeguards from the ground up.

2.3 Proposed System

The proposed cloud-based comics creation system is structured as a **web application** backed by cloud services to meet the above requirements. The architecture can be viewed as a three-tier model comprising the client interface, the server application logic (including real-time collaboration engine), and the cloud storage/database layer. Below is an overview of how the system works:

- **Client-Side Application:** The front-end will be a rich web application (accessible via browser) that provides the canvas and editing tools for users. Technologies such as HTML5 Canvas or WebGL will be used for the drawing interface to ensure smooth, vector and raster graphics editing. The UI will have panels for drawing, a toolbox (for pen, brush, color selection, shape tools), text editing for speech bubbles, and a timeline or page navigator for multi-page comics. A key component on the client side is the collaboration indicator – for instance, colored cursors or highlights show where other collaborators are working in real time. The client captures user actions (strokes drawn, text changes, object moves) and sends these as operations to the server. It also listens for updates from other collaborators to apply them locally, giving the impression of a shared workspace. To support offline-or-intermittent work, the client can queue operations if the network is lost, and synchronize when connectivity is restored.

- **Server-Side Logic and Collaboration Engine:** At the core of the system is a collab-

oration server that manages sessions for each project. When users join a project, the server authorizes them and then establishes a session where it relays updates between users. This could be implemented using WebSockets for low-latency bidirectional communication. The server uses algorithms (such as Operational Transformation or Conflict-free Replicated Data Types) to ensure that concurrent drawing or editing actions are merged consistently. For example, if two users draw on the same panel, both strokes should appear; if they edit the same text box, the transformations are applied so that no text is lost. The server also interacts with the version control subsystem: when a user saves a version or when an automatic checkpoint is triggered, the server packages the current state of the project (which may include vector graphics data, image links, text content, etc.) and stores it in the database as a new version entry. It may store differences (deltas) or full copies depending on what's more efficient. The logic also handles user management (sign-ups, logins, permission checks when someone tries to access a project) and serves the project data to clients on load. For comments and annotations, the server provides endpoints or channels to submit a comment linked to an element or coordinate on the page, and then distributes that to all viewers of the project. Additionally, the server enforces access control – ensuring that only invited collaborators can join a project's session and that they have the correct permissions (e.g., a viewer won't be allowed to send draw operations) ([7]).

- **Cloud Storage and Database:** The system utilizes cloud storage for persisting comics data. This includes a database (or a combination of databases) to store structured data like user accounts, project metadata, text content, and references to assets, as well as blob storage for large binary assets like images or PDF exports. For instance, page images or high-resolution artwork layers might be stored in an object storage service (like AWS S3 or Azure Blob Storage), whereas the layout structure (positions of panels, text content, vector stroke data) could be stored in a document-oriented database (since it's hierarchical) or a relational database with appropriate schema. The version history might be stored as a linked list of changes or snapshots in the database, with each version pointing to the necessary assets (which are themselves stored and never

deleted to preserve history). Given the need for real-time performance, the system might also use an in-memory data store or cache on the server to keep the current state of each project quickly accessible (so that new collaborators joining can get the latest state without rebuilding it from a long history). The cloud infrastructure will also handle backup and replication – ensuring that if a server fails, another can take over, and no data is lost (through regular backups of the database and storage, possibly geo-replication for disaster recovery).

- **Version Control Implementation:** The version control in this system is not exposed to the user as a complex Git-like interface, but under the hood it implements similar concepts. Each saved state gets a unique ID and timestamp. The system could allow branching by essentially duplicating a version into an alternate timeline (for advanced use cases, though initially it might enforce a linear history to simplify). On the server side, a diff algorithm might store only changes (for example, if only one panel changed between versions, the system notes that difference), which saves storage space and allows showing changes highlights. The interface for users would be a "History" slider or list where they can click on an older version and choose to restore or fork it. Upon restore, the server takes that snapshot as the new current state (possibly also keeping the newer changes in another branch or discarding them based on user choice). All of this ensures the integrity of versions – the system will not lose past data and users can trust that any prior work can be recovered.

- **Integration of AI Modules:** While the primary focus is on user-driven creation, the system design leaves room for integrating AI-assisted features in the future. For example, an AI module could offer suggestions like auto-suggesting backgrounds, or performing image enhancements, or checking the text for spelling and tone. These would be implemented as optional services that the user can invoke (e.g., a "auto-fill background" button). In the proposed design, such a module would send the image data to an AI service, receive the processed result, and then integrate it as an edit (maybe marked as AI-generated for transparency). It's important to note that any AI feature will undergo ethical review (as discussed in the next section) to prevent biases

or misuse. For the initial system, we assume minimal AI involvement aside from possibly using AI for content moderation (ensuring no explicit content is inadvertently stored publicly, depending on project policy) ([4]).

- **Client-Server Interaction Example:** To illustrate the flow: suppose an artist is drawing a line. As they drag their stylus, the client captures the stroke path in real-time and sends a stream of points to the server (perhaps throttled to a reasonable update rate). The server receives these and immediately broadcasts them to any other collaborators' clients, which then render the line progressively. Simultaneously, the server could log this action so that if a version save happens, that stroke is included. Another example: a writer types dialogue in a speech bubble. The keystrokes might update the text in real-time for others, or the system might wait until the writer finishes a text box to send the full text (to avoid excessive network chatter). In either case, other users see the updated dialogue appear. When the writer hits "save version," the server creates a new version entry with the updated text.

- **System Deployment:** The system will be deployed on a cloud platform to leverage scalability and availability. The front-end could be served as a static web app (downloadable assets like HTML/JS/CSS), while the back-end runs on cloud servers or container services. Using modern DevOps practices, updates to the system can be rolled out without downtime (e.g., using container orchestration to update one instance at a time). Data stores are cloud-managed to reduce maintenance overhead and ensure reliability (with SLAs for uptime). We will also use encryption (TLS) for all network communication. User files and data on the server side are encrypted at rest, adding a layer of security in case of any breach of the storage.

- **Backup and Recovery:** The proposed system includes automated backups of the database and user assets at regular intervals. In case of accidental deletion or catastrophic failure, these backups ensure that projects can be restored to at most the last backup point. Coupled with the version history, even if a user accidentally deletes some content, they can restore from history or an admin could recover from a backup

if needed, underscoring our commitment to data durability.

In summary, the proposed system's design harnesses cloud technology to provide a seamless collaborative experience for comic creation. By combining a responsive client-side app with a robust server managing real-time collaboration and version control, and secure cloud storage on the back-end, it meets the earlier requirements. The architecture is geared towards ensuring that multiple creatives can work together as if sharing a physical studio, while the cloud backbone handles the heavy lifting of synchronization, storage, and security. In the next section, we discuss the ethical considerations inherent in this system and how we plan to address them.

2.4 Ethics

Any platform that enables creative collaboration must contend with ethical considerations, especially when leveraging cloud infrastructure and potentially AI-driven features. Addressing these concerns is crucial to ensure the platform is fair, trustworthy, and respects the rights of its users.

Creative works are the intellectual property of the artists and writers, and hosting these works on a cloud platform raises concerns of unauthorized access, theft, or misuse. Ethically, the platform has a duty to protect users' creations with the highest standard of security.

To ensure asset security, data is encrypted end-to-end. All projects and asset files stored on the server are encrypted at rest; even if someone were to obtain the raw stored data, it would be unintelligible without the proper decryption keys. Access to projects is gated by strong authentication.

Another aspect of security is data privacy: users might chat or discuss story ideas through the platform, which should also be kept private among collaborators. We ensure that private communications or any personal data are stored securely and only accessible to intended recipients

Chapter 3: Architecture and Design

This chapter provides an overview of the system architecture and design of the Digital Comics Illustration Tool. It covers the technical structure of the platform, its deployment model using Google Cloud Platform (GCP), the guiding design principles, patterns, and data models that shape its implementation, and finally, the user interface design. The chapter outlines how each major subsystem—frontend, backend, database, and AI services—interacts to create a cohesive, scalable environment for collaborative comic creation. Emphasis is placed on how cloud-native technologies are utilized to ensure real-time responsiveness, data integrity, and ease of deployment across varied user devices.

In addition to the system structure, this chapter discusses the modular design strategies employed to enhance maintainability, extensibility, and user-centered performance. The user interface design is also elaborated upon, highlighting how React components were structured to support different creative roles with simplicity and clarity. System diagrams, including component, deployment, and data flow diagrams, are included to visually illustrate how users, services, and storage components interconnect and operate throughout a typical project workflow. Together, these elements provide a detailed understanding of the technical foundation upon which the platform was built.

3.1 System Architecture

The architecture of the platform follows a client-server model designed for scalability, real-time responsiveness, and modular development. It separates concerns across multiple layers: the frontend (React), the backend (Node.js with Express), cloud storage and databases (Firestore for real-time synchronization and MySQL for structured, relational data), and deployment infrastructure (Google Cloud Run). Each layer is independently scalable and loosely coupled, allowing the system to handle increases in user load, accommodate future feature expansions, and maintain high availability. The design emphasizes clear separation of responsibilities between the presentation, logic, and data management tiers, ensuring that updates or modifications in one layer can be made with minimal impact on the

others.

Communication between the layers is achieved primarily through secure HTTPS requests for standard interactions and WebSocket connections for real-time collaborative editing, ensuring low-latency data transfer for dynamic drawing and version updates. By adopting a cloud-native architecture, the platform benefits from automatic scaling, managed security, and minimized operational overhead, which are critical for supporting simultaneous collaboration among users across different locations. Together, these architectural choices create a robust foundation that meets both the creative and technical demands of modern digital comic production.

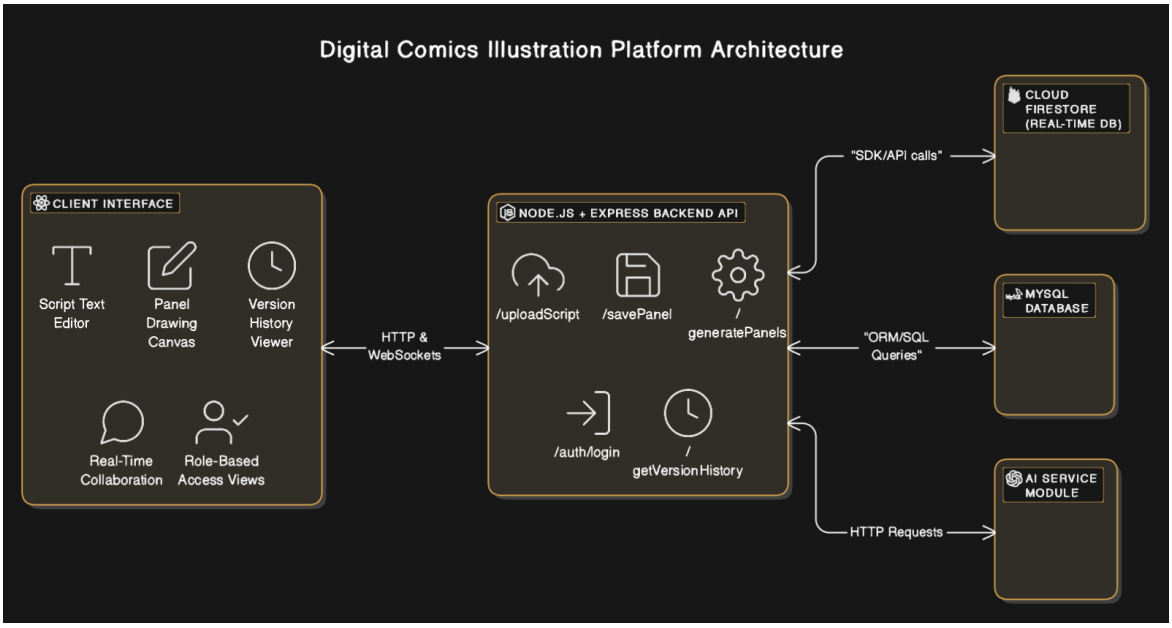

Figure 3.1: Component Diagram: To show the React frontend, backend API, Firestore, MySQL, and AI service modules.

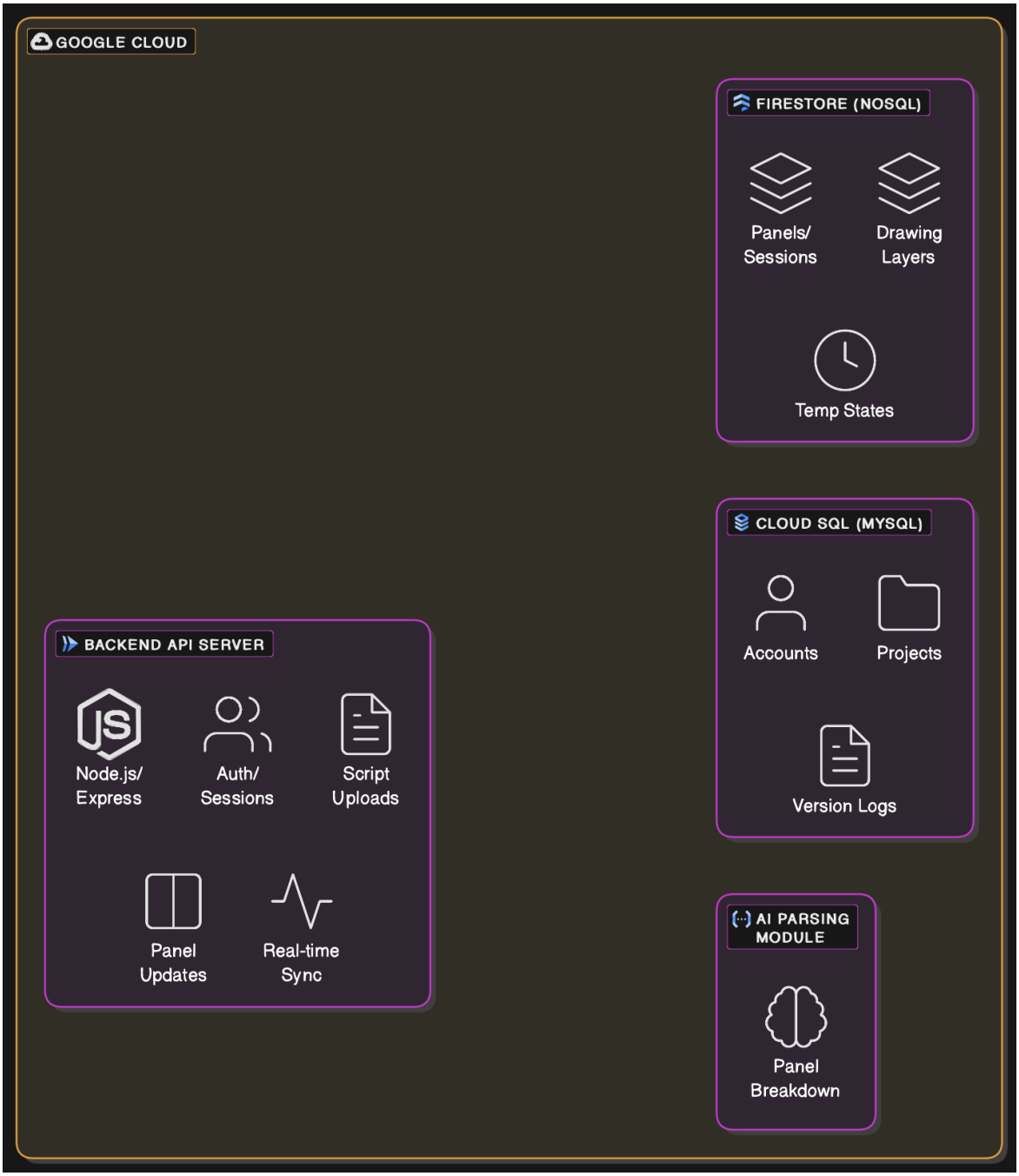

Figure 3.2: Deployment Diagram: To illustrate Cloud Run services, Firestore connections, and client-server interaction.

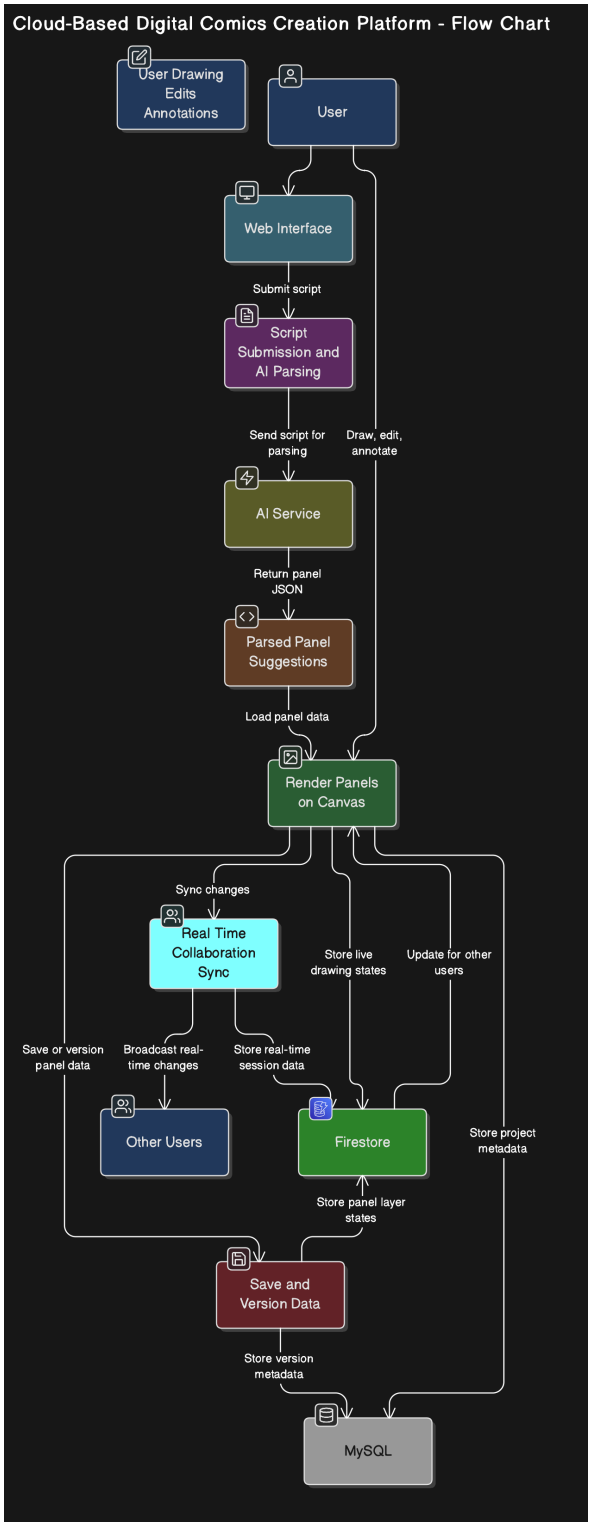

Figure 3.3: Data Flow Diagram (Level 1): Showing flow from user input to AI to canvas to backend storage.

3.1.1 Using Google Cloud Platform

The backend and AI features are deployed using Google Cloud Run, a managed compute platform that enables containerized apps to be deployed quickly and scale automatically. Cloud Run was chosen due to its compatibility with Dockerized Node.js environments, ease of deployment, and integration with other Firebase services like Firestore.

Key GCP Services Used:

- Cloud Run: Hosts the backend API and AI script processor.
- Firebase (Firestore): Provides real-time database syncing for UI updates and collaboration.
- Cloud Functions (optional): Intended for triggering events or processing AI panel suggestions in the future.
- Cloud Build: Automatically builds and deploys code from the GitHub repository.

Using GCP ensures that the platform remains highly available, responsive, and easy to maintain with minimal DevOps overhead.

3.2 System Design

The system was designed to be modular, extensible, and aligned with the collaborative workflows inherent to comic production. Its core modules — Script Processing, Panel Editor, Drawing Canvas, and Version Tracker — are each developed as independent but interoperable components, ensuring that the system can evolve without requiring significant rework across the entire codebase. By isolating functionalities into discrete modules, the platform supports a clean separation of concerns, allowing for easier maintenance, faster feature upgrades, and future scalability. This modular architecture also facilitates role-specific interactions; for example, writers engage primarily with the Script Processing and Panel Editor modules, while artists focus on the Drawing Canvas, and editors interact extensively with the Version Tracker to manage project revisions and feedback cycles.

Each module communicates with underlying cloud services through well-defined RESTful APIs, promoting seamless data flow and synchronization across different parts of the system. The Script Processing module leverages an AI service to parse uploaded text into suggested panel layouts, reducing manual work for writers. The Panel Editor enables fine-tuning of panel organization and dialogue placement, directly reflecting updates in the Drawing Canvas, which supports layered artwork workflows such as penciling, inking, and coloring. Meanwhile, the Version Tracker maintains a complete, timestamped history of changes, interfacing with both Firestore for real-time updates and MySQL for long-term archival. This layered communication between modules and cloud resources ensures that multiple collaborators can work simultaneously while maintaining system integrity, responsiveness, and project consistency.

3.2.1 Design Principles

- **Separation of Concerns:** The platform enforces a strict division between responsibilities in the frontend and backend. The React frontend focuses exclusively on managing the user interface, handling component rendering, user input events, and client-side state management. Meanwhile, the Node.js backend processes requests, enforces business logic, interacts with the databases (Firestore and MySQL), and exposes RESTful APIs for client communication. This clear boundary ensures that each layer can evolve independently, promoting better maintainability, easier debugging, and reduced system complexity.

- **Single Responsibility Principle:** Each functional unit of the system—whether a React component or a backend service—was designed to perform exactly one well-defined task. For instance, the ScriptUploader component solely manages the uploading and initial handling of comic scripts, while the DrawingCanvas component is responsible only for enabling user interactions with the illustration canvas. Similarly, backend routes and controllers are organized so that each endpoint (e.g., /uploadScript, /saveDrawing) corresponds to a single type of operation. Adhering to this principle minimizes coupling, improves code clarity, and makes future updates or

refactoring straightforward.

- **Modularity:** The system architecture emphasizes modularity across both the frontend and backend. Independent components and services are designed with clear interfaces and minimal dependencies, allowing them to be updated, replaced, or scaled individually without impacting the rest of the system. For example, the AI-assisted Script Processing module can be swapped out or enhanced without necessitating changes in the Drawing Canvas or Version Tracker modules. This modular approach accelerates feature development, simplifies testing, and lays a strong foundation for long-term system evolution.

- **Scalability:** Scalability was built into the backend architecture from the outset. The Node.js server is designed to be stateless, meaning that it does not retain session-specific information between requests. Instead, all persistent data is managed through Firestore and MySQL databases. This statelessness enables the backend to be containerized and deployed via Google Cloud Run, allowing the system to automatically scale up or down based on demand without manual intervention. As user activity increases, additional instances can be spun up rapidly, ensuring consistent performance and availability.

- **Responsiveness:** Delivering a highly responsive user experience was critical, particularly for interactions involving the Drawing Canvas and real-time collaboration features. By leveraging Firestore's real-time synchronization capabilities, the system ensures that updates—such as brush strokes, text changes, or panel rearrangements—are immediately reflected across all connected clients. This real-time propagation of data minimizes user-perceived latency and supports seamless collaborative editing, essential for maintaining creative flow in a multi-user environment.

3.2.2 Design Pattern

The project follows the Model-View-Controller (MVC) design pattern:

- Model: Firestore and MySQL store the application's state and user data.

- View: The React components render the user interface.
- Controller: Express backend routes handle input, run business logic, and interact with the database.

React's component-based structure naturally supports the View layer, while Express acts as the Controller, and Firestore/MySQL form the Model tier. This pattern ensures logical separation and maintainability.

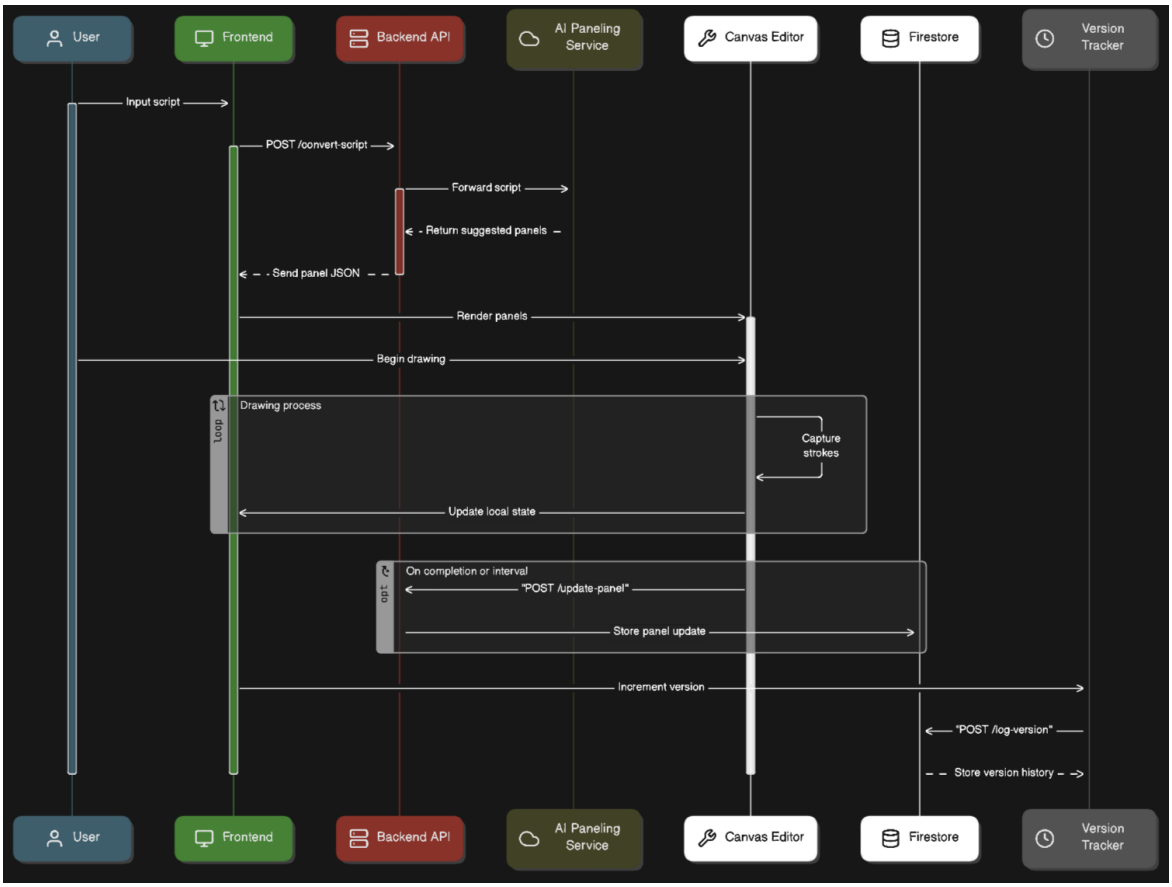

Figure 3.4: MVC Layered Diagram: Visual representation of data movement between model, view, and controller.

3.2.3 Data Model

Two primary data models support this system:

3.2.3.1 Firestore (NoSQL for real-time updates)

- Collection: `projects`

- Fields: `projectId`, `title`, `createdAt`, `updatedAt`

- Collection: `panels`
  - Fields: `projectId`, `projectId`, `text`, `coloredLayer`

- Collection: `versions`
  - Fields: `versionId`, `timestamp`, `projectId`

3.2.3.2 MySQL (Structured storage for tracking and relationships)

- Table: `Users`
  - Fields: `userId`, `name`, `role`, `email`

- Table: `PanelChanges`
  - Fields: `changeId`, `panelId`, `layerChanged`, `editorName`

3.3 User Interface Design

The user interface was designed with simplicity and clarity in mind, especially given the complexity of comic production workflows. Each user action is mapped to a dedicated component in React, ensuring consistency and reusability.

Main UI components

- ScriptUploader: Text area for writers to input and submit their script.
- ComicPanelEditor: Core UI area showing all panels with editable fields.
- DrawingCanvas: Embedded within each panel to support pencil, ink, and color sketches.
- Dashboard: Summary of version history and last update timestamps.
- VersionHistoryList: (Planned) Shows list of past versions with options to restore.

UI Design Considerations:

- Clean grid layout to avoid clutter

- Consistent typography and icons for intuitive navigation
- Responsive design that adjusts to desktop or tablet use
- Visual separation between roles using color tags (future feature)

Figure 3.5: Component Hierarchy Diagram: To show parent-child relationships between React components.

INCLUDE WIREFRAMES HERE

Chapter 4: Implementation

This chapter describes the concrete steps taken to develop the Digital Comics Illustration Tool, including the tools used, the software development methodology applied, the code structure, major implementation modules, and the process of integrating and deploying the system. The implementation process follows the design principles outlined in Chapter 3, translating them into working software that fulfills the project's objectives.

4.1 Software, Tools and Technologies

The development of this project leveraged a range of modern tools and frameworks across the frontend, backend, database, and deployment layers. The chosen stack prioritizes modularity, scalability, real-time responsiveness, and ease of use.

4.1.1 Frontend Tools

- React.js: Chosen for its component-based architecture and robust ecosystem. Supports real-time UI updates and responsive design.
- JavaScript (ES6+): Main language for frontend logic and interactions.
- Axios: Used for making HTTP requests to the backend API.
- HTML5 Canvas API: Powers the drawing interface embedded within each comic panel.
- CSS Modules / Inline Styling: Used for styling components.

4.1.2 Backend Tools

- Node.js: Runtime environment for JavaScript server-side code.
- Express.js: Lightweight web framework used to build RESTful APIs for the system.
- Docker: Used to containerize the backend for deployment on Google Cloud Run.

4.1.3 Database Tools

- Firestore (Firebase): Used for real-time syncing of UI components and storing project/panel data.
- MySQL: Used for structured data like version history, user activity logs, and change tracking.

4.2 Software Development Approach

The project followed an agile-inspired iterative approach, though conducted individually. Features were broken into milestones and implemented in sprints over several weeks. Each sprint focused on a single area of functionality, such as:

- Sprint 1: React UI setup and script input
- Sprint 2: Backend endpoints for script conversion and panel updates
- Sprint 3: Canvas drawing functionality and state handling
- Sprint 4: Version tracking and dashboard integration
- Sprint 5: Code refactoring, local testing, and deployment prep

4.3 Implementation Details

The following is the file structure used for the root code:

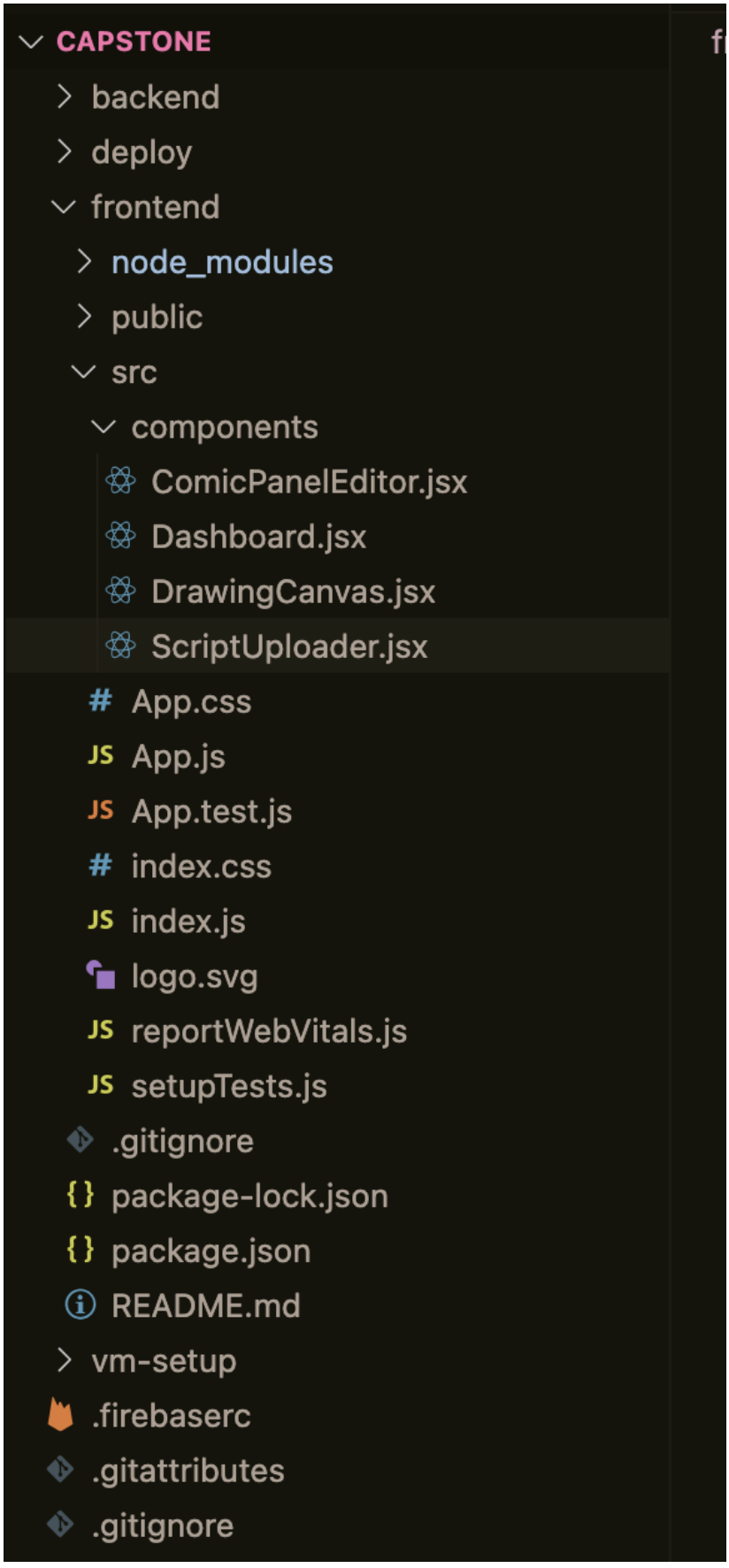

Figure 4.6: File structure for root code.

4.4 User Interface Design

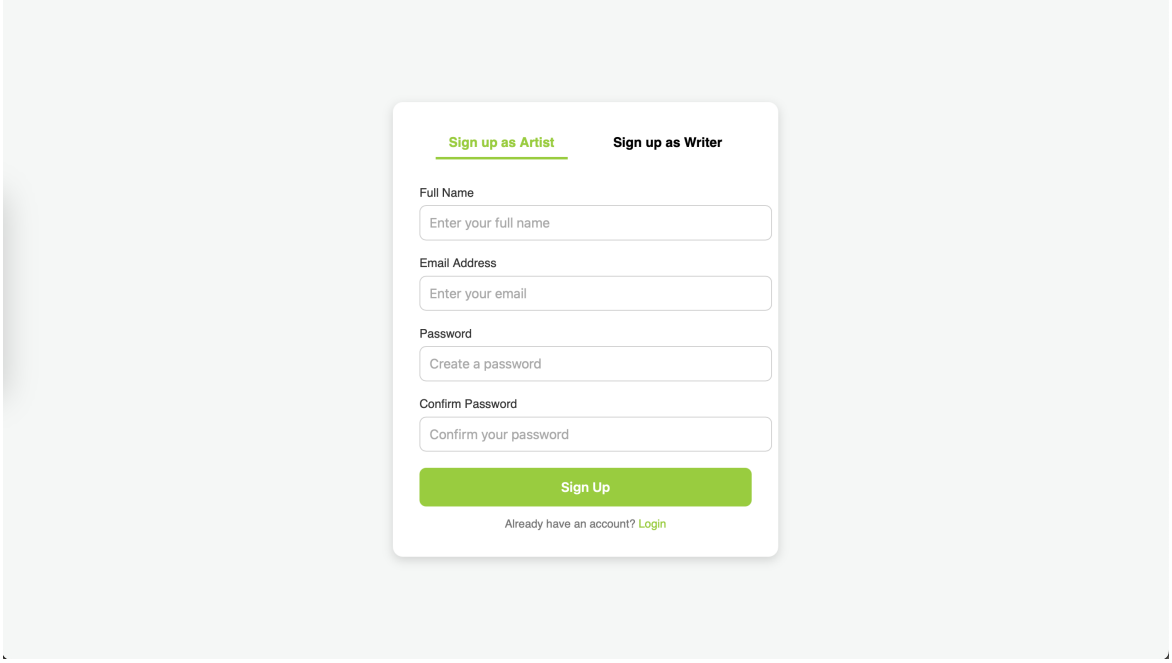

Figure 4.7: Sign up page.

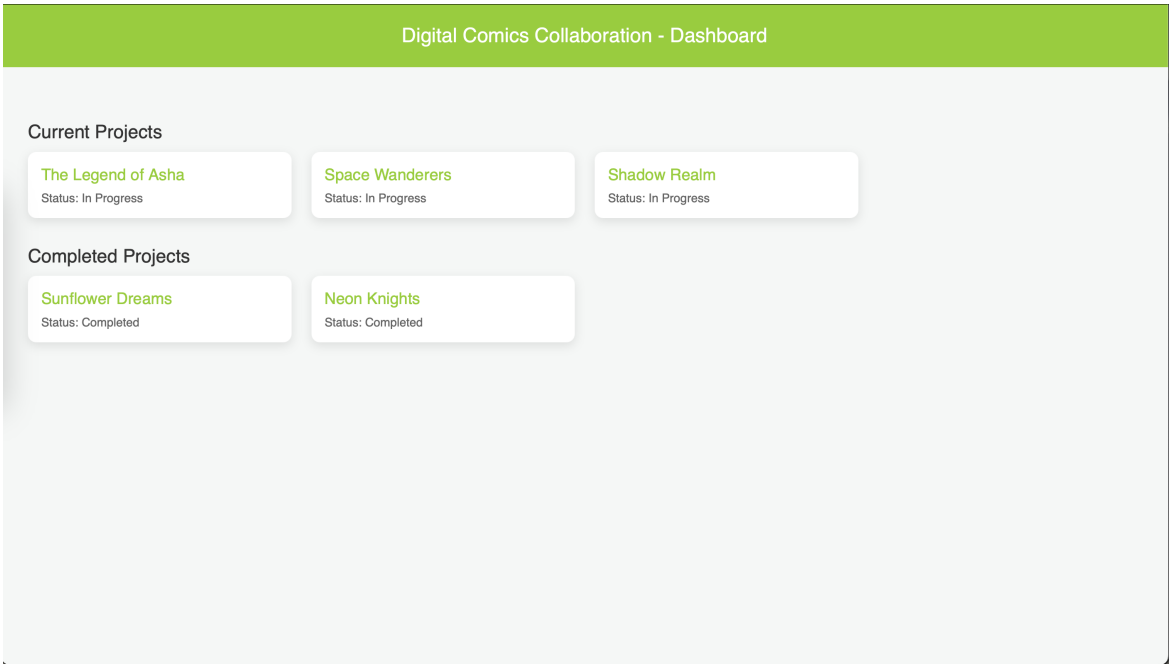

Figure 4.8: Home page displaying projects.

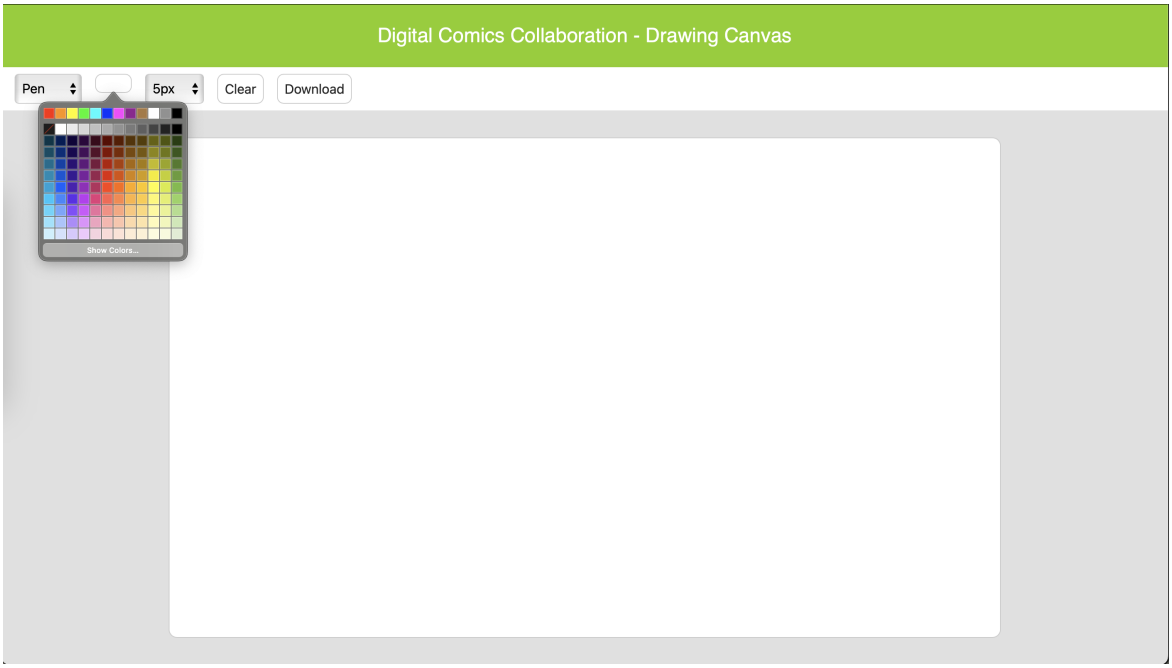

Figure 4.9: Canvas page for drawing.

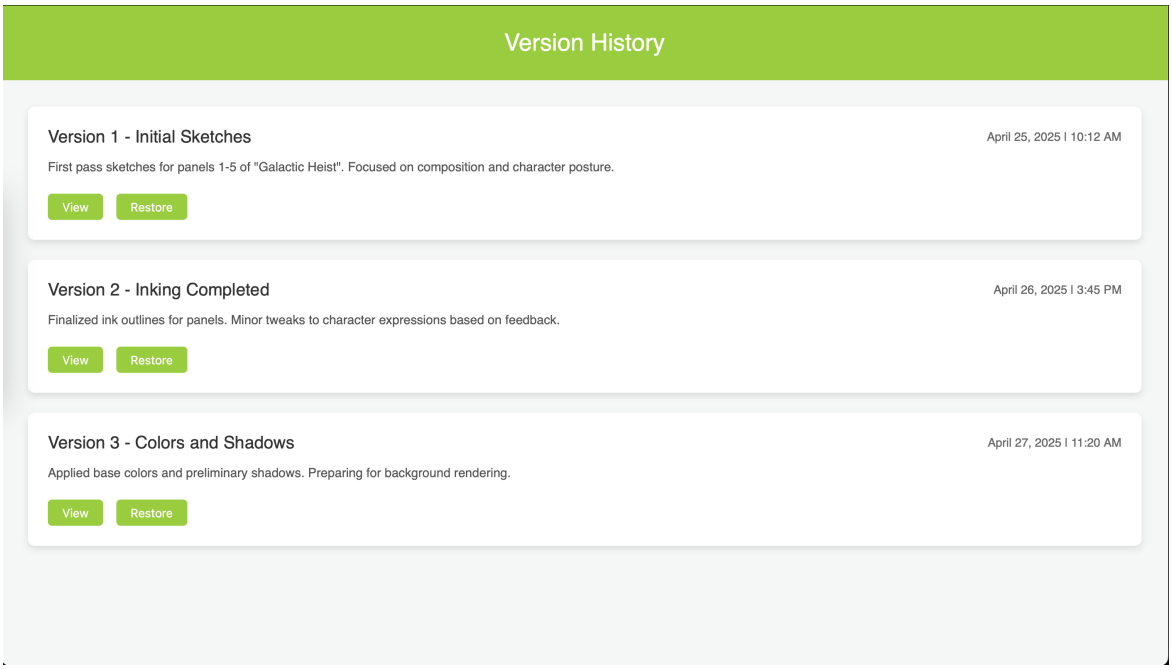

Figure 4.10: Page for viewing version history.

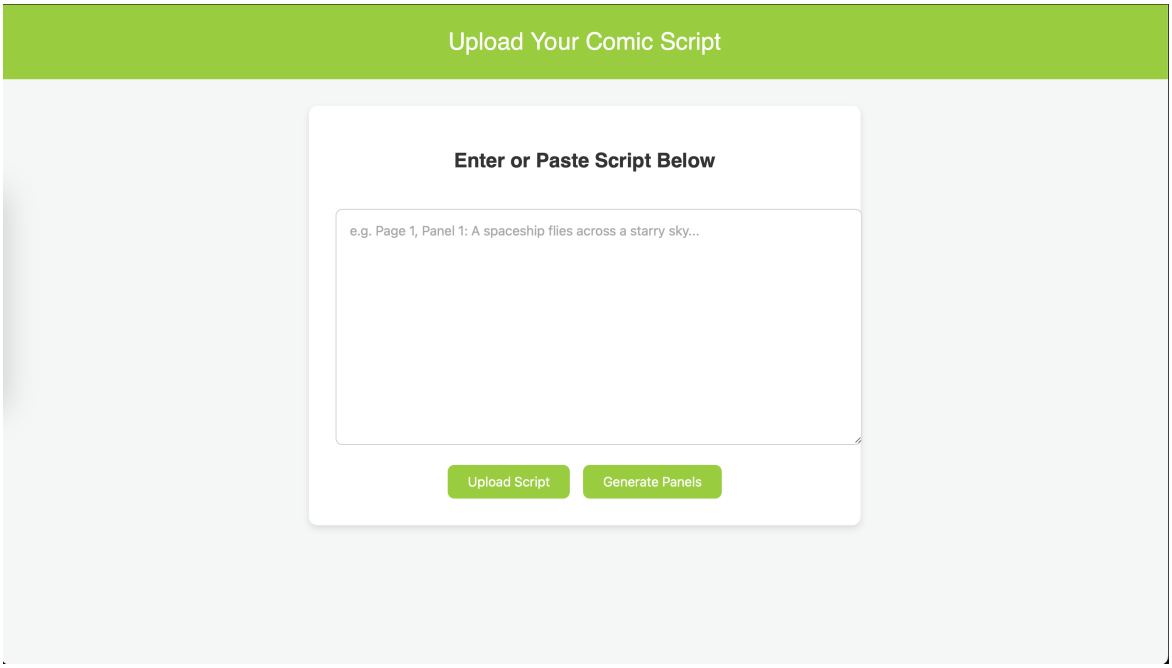

Figure 4.11: Upload page for scripts.

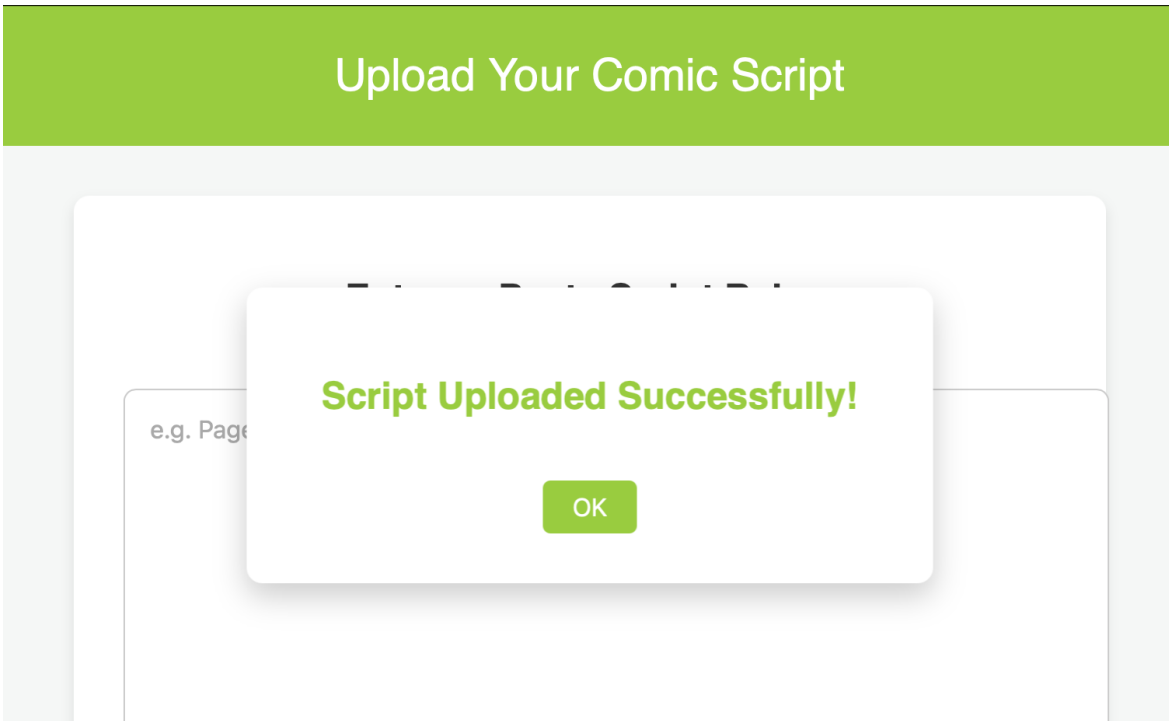

Figure 4.12: Script uploaded successfully.

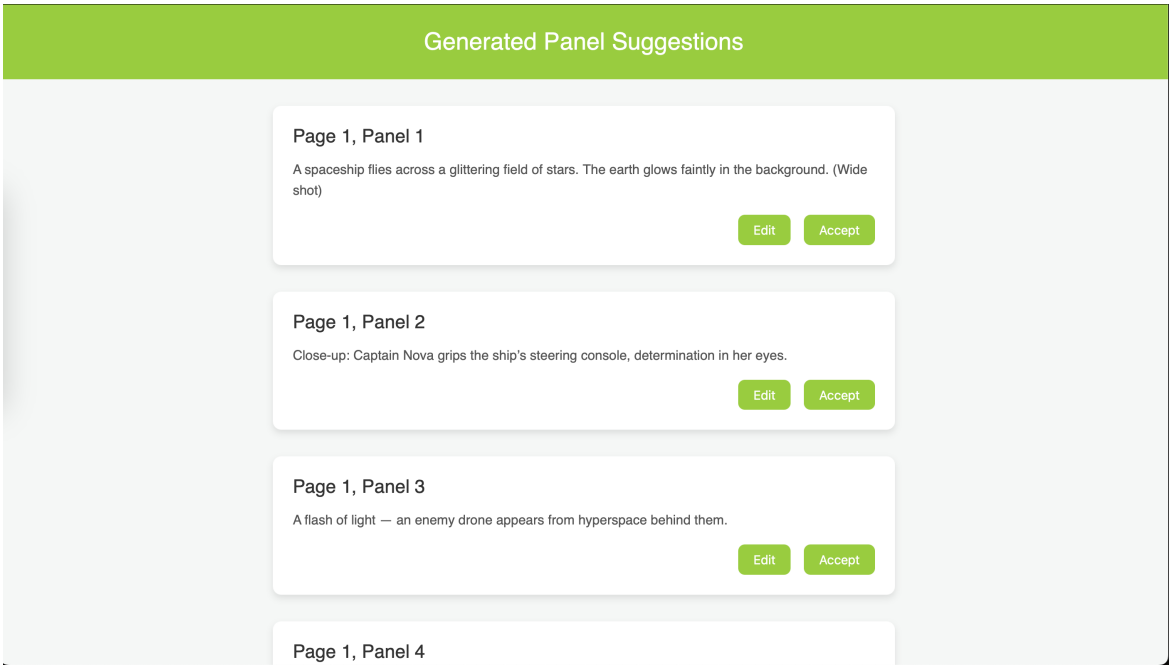

Figure 4.13: Accepting or rejecting AI generated panel separation.

4.5 Deployment Process

The system is deployed using Google Cloud Run and GitHub-integrated continuous delivery.

Deployment Steps:

1. Backend code containerized via Docker
2. Built using Google Cloud Build
3. Deployed to a Cloud Run service instance
4. Service configured for unauthenticated access (for prototype testing)
5. Frontend hosted locally or optionally deployable to Firebase Hosting Environment variables are managed through .env files and secrets in Cloud Run. The full system can be restarted or updated via push-to-deploy workflows on GitHub.

Chapter 5: Testing and Results

For this capstone project, a range of testing methods were applied to validate the functionality, integration, usability, and accessibility of the Digital Comics Illustration Tool.

Testing activities focused primarily on four areas: unit testing of individual modules, integration testing across system boundaries, manual exploratory testing by simulated users representing different stakeholder roles, and browser compatibility testing to ensure cross-platform performance. Due to project time constraints, full automated system testing and formalized usability surveys have been deferred for future development phases.

This chapter documents the testing approaches taken, summarizes the outcomes, and discusses insights gained that will inform future improvements.

5.1 Unit Testing

Unit testing involved verifying that isolated pieces of the system — including frontend components and backend routes — performed correctly when subjected to controlled inputs and conditions. This form of testing was essential for catching low-level errors early and ensuring that building blocks of the application were reliable.

Components Tested

- **ScriptUploader Component:** Confirmed that the input field captured user text correctly, maintained internal state using useState, and triggered Axios POST requests appropriately upon submission.
- **ComicPanelEditor Component:** Verified that panel data passed from parent to child components was accurately rendered, editable fields updated state correctly, and edits could be propagated back to the backend.
- **DrawingCanvas Component:** Tested that mouse event handlers (onMouseDown, onMouseMove, onMouseUp) initiated and tracked drawing paths correctly without error across multiple draw-erase cycles.

- **Backend /convert-script Route:** Sent sample script inputs and confirmed that the panelization logic returned arrays formatted according to API contract (e.g., array of panel objects with text).
- **Backend /update-panel Route:** Submitted simulated panel updates and verified backend acceptance, successful database storage, and appropriate success response codes (HTTP 200).

The testing approach involved manual assertion through console logging, API testing using Postman, and the use of temporary in-component test states.

5.2 Integration Testing

Integration testing focused on validating the flow of data and commands across different parts of the system: frontend, backend, cloud services, and database layers. This ensured that modules worked together cohesively and reliably.

Integration Flows Tested:

- Frontend to Backend API:
  - Verified successful HTTP POST request from the React frontend to the Node.js Express backend when uploading scripts.
  - Checked the return of correctly structured JSON panel data for frontend rendering.
- Canvas Drawing to Backend Update:
  - Confirmed that when an artist drew on a panel canvas and clicked save, the resulting image data or metadata was sent correctly to backend endpoints and persisted in Firestore.
- Version History Integration:
  - Validated that after each save event, the Dashboard module updated with a new version count and corresponding timestamp.

- Frontend State Consistency:
  - Checked that React component state (useState, useEffect) updated predictably following backend confirmation of saved changes.

The testing strategy consisted of manual testing of user workflows combined with the observation of real-time data propagation using Chrome Developer Tools, specifically the network tab and state monitors. The results showed robust integration with minimal lag or error propagation across system boundaries.

5.3 Manual Testing

Manual testing was conducted to simulate end-user workflows and detect issues that might not surface in structured unit or integration tests. It was crucial for gaining subjective insights into the usability and flow of the application.

5.3.1 Procedure

Potential users — acting as writers, artists, and editors — performed a series of exploratory tasks under informal observation:

- Upload scripts with varying lengths and complexity.
- Review and edit panel layouts generated by the AI module.
- Draw sketches on panel canvases at different levels of detail.
- Save multiple versions and navigate version history.

5.3.2 Key Observations

During manual testing sessions, several key observations were made regarding user interactions with the platform. Participants quickly grasped the core workflow without requiring additional guidance, indicating that the platform's user onboarding process is intuitive and accessible.

When working with script inputs, simple and straightforward scripts were processed effectively by the AI panel generation module. However, more complex scripts occasionally required manual tweaking of the panel text to achieve the desired level of clarity and segmentation.

The drawing canvas received particularly positive feedback, with users praising its immediate responsiveness and ease of use. Nonetheless, participants suggested future enhancements such as the ability to select brush colors, adjust line thickness, and implement an undo functionality to improve the drawing experience further.

Finally, the version tracking system was seen as a valuable feature for monitoring project progress and identifying specific save points. Participants recommended that future versions of the platform allow users to label versions with descriptive names (such as "initial sketch" or "colored version") to improve clarity and navigation through the revision history.

5.4 Browser Compatibility Testing

Browser compatibility testing focused exclusively on Google Chrome, Safari, and Microsoft Edge, as these are among the most commonly used browsers by creative professionals. Testing aimed to ensure that core functionalities such as script uploading, panel generation, canvas drawing, version saving, and general UI navigation performed consistently across these platforms.

Each browser was used to complete the key workflows of the system. In all cases, script uploads, API interactions, and user interface navigation behaved as expected, with no significant delays or rendering issues. Canvas drawing functionality was smoothest in Google Chrome, with Safari demonstrating slight but non-disruptive latency during rapid sketching sessions. Minor layout inconsistencies observed in Safari were addressed through targeted CSS adjustments.

Overall, the testing confirmed that the application is stable and functional in Chrome, Safari, and Microsoft Edge. No critical browser-specific issues were encountered, and users on these platforms can expect a reliable and consistent experience. Other browsers such as Mozilla Firefox were not included in this phase of testing.

Chapter 6: Conclusion and Recommendations

This project set out to address a critical gap in the comic book creation workflow by designing and implementing a centralized, cloud-hosted digital platform for real-time collaboration among writers, artists, and editors. The resulting application successfully integrates a suite of features tailored specifically to the needs of comic production teams.

Throughout the development process, the project demonstrated the feasibility of unifying the traditionally fragmented comic creation pipeline into a seamless digital environment. Writers can upload scripts and instantly receive panel breakdowns; artists can collaborate across penciling, inking, and coloring stages in real time; and managers or editors can track project progression through an integrated version history dashboard. By leveraging technologies such as React, Node.js, Express, Firestore, MySQL, and Cloud Run, the system balances real-time interactivity with robust backend processing and scalability.

Key achievements of this project include:

- Successful deployment of a containerized backend to Google Cloud Run.
- Full integration of an AI-based script analysis service.
- Implementation of per-panel drawing canvases using the HTML5 Canvas API.
- Real-time update propagation through Firestore and frontend state management.
- Persistent version tracking for historical review and potential restoration.
- Modular and scalable architecture following the Model-View-Controller (MVC) design pattern.

This project not only demonstrates technical proficiency across the full software development lifecycle — from requirement analysis to deployment — but also contributes an innovative tool concept that has clear applicability in creative industries increasingly moving towards remote and collaborative workflows.

6.1 Recommendations

Based on the experience gained during the development of this project and the analysis of its current capabilities and limitations, the following recommendations are offered for future development teams or stakeholders interested in continuing the evolution of this platform:

Adopt a Microservices Architecture for Scalability

As new features such as live chat, detailed asset management, and collaborative editing are added, the backend should transition toward a microservices architecture. This will enable independent scaling of services (e.g., canvas management, AI processing) and improve maintainability.

6.1.1 Strengthen Data Security and User Privacy

As user authentication and project sharing features are introduced, ensuring data security and protecting intellectual property must become a priority. Best practices such as end-to-end encryption for stored data, secure authentication protocols (OAuth2, JWT), and regular security audits are recommended.

6.1.2 Invest in UX/UI Enhancements

Although functional, the current user interface can be further refined to enhance usability, accessibility, and aesthetic appeal. Conducting usability testing with real artists and writers would help inform iterative improvements.

6.1.3 Integrate Third-Party Creative Tools

Future versions could consider integrating APIs from popular creative suites (e.g., Adobe Creative Cloud, Clip Studio Paint) to allow easy import/export of assets and pages, making the platform more attractive to professional users.

6.1.4 Formalize Testing and Quality Assurance Processes

While manual testing has validated core functionalities, future development should include comprehensive automated testing — including unit, integration, and end-to-end tests — to ensure system reliability as complexity grows.

6.1.5 Pursue Real-World Pilot Testing

Once the basic multi-user capabilities are stable, a pilot program involving a small group of professional or indie comic creators could provide valuable feedback, uncover usability issues, and validate the platform's effectiveness in a real-world setting.

6.2 Future Work

Although the application meets its minimum viable product (MVP) goals, several enhancements have been identified that would elevate the platform into a more production-ready state:

6.2.1 Multi-User Authentication and Role Management

Future versions should implement authentication and authorization systems that allow writers, pencillers, inkers, colorists, and editors to access role-specific tools and privileges.

6.2.2 Live Collaboration Features

Adding real-time collaboration elements such as shared cursors, instant messaging, commenting on panels, and concurrent editing notifications would enhance teamwork dynamics.

6.2.3 Canvas Export and Publishing

Development of a compositing engine that allows for the export of completed comic issues into formats such as PDF, PNG, or JPEG would make the tool more versatile for publishing needs.

6.2.4 AI Layout Enhancement

Integration of more advanced natural language processing models could provide richer panel layout suggestions, capturing nuance, scene dynamics, and emotional tone.

6.2.5 Asset Management System

A feature to store, manage, and reuse commonly used assets such as character models, backgrounds, and props could significantly accelerate the illustration process.

6.2.6 Scalability and Performance Optimization

As usage scales, optimizations around backend architecture, database indexing, and WebSocket-based real-time communication will become essential.

In conclusion, the Digital Comics Illustration Tool has established a strong foundation for revolutionizing the comic creation workflow. By continuing to refine, secure, and expand its features, it holds significant potential to empower storytellers and artists around the world to collaborate more efficiently and creatively in the digital age.

References

[1] Belda-Medina, J. Inclusive education through digital comic creation in higher learning environments. *Social Sciences 13*, 5 (2024), 272.

[2] Benatti, F. *Innovations in Digital Comics: A Popular Revolution*. Cambridge University Press, Cambridge, UK, 2024.

[3] Cahyaningrum, Y., and Wijaya, M. R. P. Digital transformation in the arts field: Creating new collaborations in the digital arts world. *Smart International Management Journal 1*, 2 (2024), 1–8.

[4] Chen, Y.-C., and Jhala, A. Collaborative comic generation: Integrating visual narrative theories with ai models for enhanced creativity. In *Proceedings of the International Workshop on Artificial Intelligence and Creativity (CREAI 2024 at ECAI)* (Santiago de Compostela, Spain, 2024), pp. 1–8.

[5] Dawood, M., Tu, S., Xiao, C., Alasmary, H., Waqas, M., and Ur Rehman, S. Cyberattacks and security of cloud computing: A complete guideline. *Symmetry 15*, 11 (2023), 1981.

[6] Gabino-Campos, M., Baile, J. I., and Padilla-Martínez, A. Social biases in ai-generated creative texts: A mixed-methods approach in the spanish context. *Social Sciences 14*, 3 (2025), 170.

[7] GmbH, F. The state of creative collaboration in 2023, 2023. Industry Report.

[8] Kumar, I. K., Shen, J., Ferguson, C., and Picard, R. W. Connecting through comics: Design and evaluation of cube, an arts-based digital platform for trauma-impacted youth. *Proceedings of ACM Human-Computer Interaction (CSCW) 9*, 2 (April 2025), Article CSCW051, 1–22.

[9] Mencarini, E., Schiavo, G., Cappelletti, A., Stock, O., and Zancanaro, M. Assessing a collaborative application for comic strips composition. In *Proceedings of INTERACT*

2015: IFIP Conference on Human-Computer Interaction (Bamberg, Germany, 2015), Springer, LNCS 9297, pp. 73–80.

[10] Sterman, S., Nicholas, M. J., and Paulos, E. Towards creative version control. *Proceedings of ACM Human-Computer Interaction (CSCW) 6*, CSCW2 (November 2022), Article 336, 1–25.

[11] The Drum, and Adobe. Creativity, collaboration and culture: The impact of working from home on marketers and marketing, June 2021. White Paper, Adobe Systems Inc.

[12] Zhang, L., Agrawal, A., Oney, S., and Guo, A. Vrgit: A version control system for collaborative content creation in virtual reality. In *Proceedings of CHI* (2023).