Development of a Child-Centric Interactive Storytelling Application

By

Loh Shin Yee

A REPORT

SUBMITTED TO

Universiti Tunku Abdul Rahman

in partial fulfillment of the requirements

for the degree of

BACHELOR OF COMPUTER SCIENCE (HONOURS)
Faculty of Information and Communication Technology
(Kampar Campus)

FEB 2026

COPYRIGHT STATEMENT

This Final Year Project report is submitted in partial fulfillment of the requirements for the degree of Bachelor of Computer Science (Honours) at Universiti Tunku Abdul Rahman (UTAR). This Final Year Project report represents the work of the author, except where due acknowledgment has been made in the text. No part of this Final Year Project report may be reproduced, stored, or transmitted in any form or by any means, whether electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the author or UTAR, in accordance with UTAR's Intellectual Property Policy.

ACKNOWLEDGEMENTS

I would like to express my sincere thanks and appreciation to my supervisor, Dr Tan Joi San, and my moderator, Puan Fatin Natasya binti Shuhaimi for the guidance, support, and valuable insights throughout the development of this project, “Development of a Child-Centric Interactive Storytelling Application”. Their expertise and encouragement have been invaluable in shaping this project. Again, a million thanks to my supervisor and moderator.

I would also like to extend my heartfelt gratitude to my family for their unwavering love, support and understanding. Their continuous encouragement has been a source of strength throughout my studies.

A special thank you goes to my friends and peers who have been supportive and understanding during this challenging process. Their valuable suggestions and moral support have kept me motivated and inspired.

Finally, I would like to thank all those who have contributed to my personal and academic growth, whether directly or indirectly, for their support and motivation.

ABSTRACT

Storytelling is an essential growing component of early childhood education, which promotes creativity, imagination, fostering language skills, and emotional expression. Yet children's access to mobile media is now near universal. For example, UK data show that by age 11 roughly nine in ten children own a mobile phone, while many apps mainly encourage watching and playing games rather than making their own content, such as safe collaboration and creative authorship. In this project, an application, Story Craft is proposed to solve this gap by offering an interactive storytelling app where children can draw, upload those drawings and select photos from the gallery to generate a story. Story Craft supports a drawing module which enables users to draw their own creative design in the drawing part and upload their drawings to generate a story for them. It also provides an uploading photo module which allows users to upload their photos from their gallery and generate a story for them. Users can select their preferred story genres and edit the generated story. The users can enhance their drawings using drawing tools with optional enhanced illustrations to become more creative and artistic. The app supports text-to-speech features which allow users to convert their generated text stories to voice. The secure login authentication module with email and password is also provided in Story Craft. The app provides a community section where users can upload their generated stories and interact through likes, shares, and comments with the users inside the community. Story Craft supports history modules which allow users to revisit and track their previous stories. Story Craft is developed using Visual Studio Code with Flutter, which is a mobile application to ensure broad accessibility across Android. The Story Craft is designed to be safe for young users. By combining drawing, storytelling, and community interaction in an application can enhance user's creativity and digital expression in a meaningful way.

Area of Study: Early Childhood Education, Mobile Application Development

Keywords: Storytelling, Interactive Storytelling, Community Interaction, Flutter, Drawing Module, Text-to-Speech, Child Safety, Creativity, Digital Expression

Table of Contents

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

API	Application Programming Interface
CRUD	Cread, Read, Update, Delete
CSS	Cascading Style Sheets
DBMS	Database Management System
FYP	Final Year Project
GPU	Graphics Processing Unit
HTML	Hypertext Markup Language
JSON	JavaScript Object Notation
RAM	Random Access Memory
SDK	Software Development Kit
SDLC	Software Development Life Cycle
TTS	Text-to-Speech
UI	User Interface

CHAPTER 1 INTRODUCTION

1.1 Background and Motivation

Storytelling is an essential component of early childhood education and plays an important role in child development. Storytelling enables early language buildings, enhances imagination, and supports emotional growth of children. Children naturally tell their stories verbally through pictures, or by just acting out scenes. These forms of self-expression support learning by enhancing their imagination and strengthening memory retention. In recent 21st century learning, with the rapid advancements in mobile technology, digital technologies have given a new chance for children to gain these experiences through mobile applications. With the growing use of tablets and smartphones in schools and homes, children nowadays have become more common in using and interacting with these educational applications.

Many platforms aim to support storytelling, drawing, or literacy, but they frequently restrict children to passive experiences, such as choosing from predefined characters, completing the fixed story templates, and reading only pre-written story books. These platforms limit the children's ability to create their own stories based on their creativity and imagination. Children prefer to communicate through either pictures or basic drawings, especially those who are still learning to read and write. However, storytelling applications that are designed for children still have limitations in terms of converting visual expression to customized digital stories. In addition, the drawing features on mobile devices do not have enhancement features to improve their drawings. The lack of visual enhancement feature will reduce creative confidence and limit the drawing function as the foundation for a story. As the research suggests, creative learning environments should offer not only freedom of expression but also responsive tools that help children to develop their ideas with guidance and support [1]. Also, the equally important thing is the social aspect of storytelling. Many existing apps do not support children to share their creations, view others' stories, and interact in commenting ways.

The project proposes the development of a mobile application that addresses these key limitations by offering a child-centric storytelling experience called Story Craft. The app allows children to draw or upload an image, generate a story based on that image, choose the genre of the story, such as adventure, mystery, or fairytale, and edit the text stories as they want. Besides, the children can choose to share their work, interact safely through likes, comments, or

reactions in a story-sharing community, which encourages collaboration. The application will be developed using Flutter, a modern cross-platform development framework, to ensure broad accessibility across Android. The system will be designed with a child-friendly interface, simplified navigation, and parental or teacher oversight features to ensure a secure user experience.

The main motivation of this project is to bridge these gaps by developing StoryCraft, a child-centric storytelling mobile application that combines visual creativity, AI-assisted story generation, editing flexibility, and safe social interaction. The app allows children to draw or upload images, generate genre-based stories, edit content, and share in a moderated community. By integrating these features in one platform, the project aims to improve engagement, creative confidence, communication skills, and collaborative learning while maintaining child safety and usability.

1.2 Problem Statements

The existing storytelling applications often rely on pre-defined templates that restrict a child's natural visual imagination and lack of enhancement tools needed to turn simple sketches into polished narratives. Furthermore, the absence of safe and collaborative communities prevents children from sharing their work and learning through social interaction.

i. Lack of personalized and creative story generation for storytelling

Most traditional digital storytelling applications only provide fixed stories, drag-and drop blocks, and pre-defined templates with minimal opportunity for children to add their own ideas. While these tools can be useful for structured learning, they restrict imagination and do not allow children to create stories that reflect their own ideas and imagination. Younger children, especially between the ages of 5 to 10, often think in pictures rather than words and would benefit more from image based or visual storytelling features as Piaget's theory of cognitive development suggests that children in the preoperational stage primarily use symbolic thinking in pictures before they develop the ability to think abstractly with words [14]. However, current apps rarely allow children to generate their own stories from drawings, select genres, and edit their generated stories. This limitation reduces personalization and prevents children from exploring their creativity. Without these features, children may lose their interest and feel

disconnected from the stories, missing opportunities for engagement, self-expression, and ownership over their learning process.

ii. No drawing assistance and visual enhancement

Drawing is an essential and natural way for children to express their own thoughts and feelings, especially for those who are still learning to read and write. Yet, many existing storytelling apps treat drawing only as a basic and static function, without providing tools to enhance and improve children's work. Young children often struggle to accurately represent their ideas visually and lead to unclear shapes, messy sketches, and incomplete details. Without guidance and visual enhancement assistance, this can result in frustration and discourage them from drawing. Current apps also lack features to refine and improve children's sketches, such as improving line clarity and coloring neatly. By not supporting visual enhancements, children miss the chance to build confidence in their artistic skills and see their imagination come to life. Furthermore, the absence of link between visual input and story generation limits the opportunity to connect drawing with storytelling, reducing both creative expression and learning value.

iii. Absence of collaborative and community features

Another main problem in existing storytelling and education applications designed for children is the absence of safety and child-friendly community features. Most platforms restrict children's creations to local storage or private viewing, preventing them from sharing their work with each other, receiving feedback, and learning from one another creativity to improve their imagination skills. The lack of collaborative community features reduce motivation and deprives children of opportunities to learn through social interaction. According to Vygotsky's social development theory, children always acquire skills and knowledge not only through individual activity, but also through interaction with peers [2]. Without community features such as likes, commenting, and sharing stories, children's creative work often goes unrecognized, which can lessen their enthusiasm for creating more. In addition, the absence of collaborative spaces limits opportunities for children to gain inspiration from others' stories and to practice positive online interaction at an early age. Incorporating safe community engagement would not only foster creativity but also encourage teamwork, critical thinking, and digital literacy.

1.3 Project Objectives

In this project, the main objective is to develop a mobile application that can empower children to create personalized stories through drawings or insert images with features that support story generation, visual assistance, and a safe storytelling community for sharing and interaction.

i. To implement a child-friendly drawing module with visual assistance tools to support creative expression and ease of use

Due to the absence of visual enhancement features in many creative storytelling apps, the objective is to focus on creating an intuitive drawing interface for those children who may struggle with freehand drawing. The implementation of the drawing module will enhance cognitive and motor skill levels of children. The children can draw what they want, and ideas based on their imaginations. Lack of drawing modules with visual assistance tools affect children's drawing experience and limit their imagination. To address the problems, the proposed application, Story Craft, will include a visually interactive drawing module with features drag-and-drop and coloring zones. These enhancements help children to develop their creative abilities in an enjoyable manner. The interface will use large buttons and bright visuals for children to increase their usability expectations. By providing this support, the app becomes more accessible and encourages children to explore their creativity.

ii. To develop a personalized and creative storytelling feature that generates editable stories based on user-provided drawings or uploaded images

One of the main objectives of this storytelling project is to provide creativity among children by allowing them to generate a story that is directly inspired by their own drawings or images. This objective aims to address the problems of lacking personalized and creative storytelling opportunities in current applications, which rely on static and pre-written content. Story Craft app will allow children to input their unique visual ideas by integrating image-to-story based features into basic storytelling app. Whether applying in-app drawing or uploading images from their gallery, all these actions are transformed into narrative form story. In order to guide the storytelling experience, children can choose from different kinds of story genres, such as mystery, fairy tale, or adventure, which helps them to explore various themes. Besides, the generated story is not fixed, the children are able to edit and expand the generated story into their preferred content. These kinds of interactive actions not only can foster imagination and creativity of children but also enhance their language skills.

iii. To build a safe and moderate community platform for children to share, like, and comment on stories

The third objective of the storytelling project is to foster collaboration, social engagement, and learn from each other through a community module within the app. The safe and moderated community module addresses the problem of lacking collaborative and interactive spaces for children especially for those current storytelling apps. The users can only see their own work themselves without being given chances to communicate with other users' ideas. In Story Craft applications, the children are allowed to share their generated story and drawing to a shared community, where they can also read, view other's content, and interact with stories created by others through moderated comments in a controlled environment. All the comments will be filtered out through the system involving keyword detection and restricted comment options. The interactive community module aims to encourage positive feedback and communication among children and further enhance their user experience. Safety mechanisms like moderation, filtering, and review workflows are includes to support positive and age-appropriate participation.

1.4 Project Scopes

The scope of Story Craft is to develop a mobile application that allows children to create their personalized stories through drawings or uploaded photos. Children will be able to draw directly within the app or upload a photo, select a preferred story genre, and receive a generated story on the voice based. The children can edit their generated story manually based on their imagination. The app will also provide enhanced tools for children to improve their drawings. Each child can have their personal account by providing a user authentication module for them to log in and log out. Also, the children can communicate and share their stories through a community module.

i. Login module

The login module is responsible for managing secure access to the application by enabling users to register, log in and log out. The user authentication module ensures that each child has a unique profile that stores their personal data, including saved stories and interaction history. By creating an individual account, the application can provide a secure account-based personalization and controlled environment suitable for children and enhance their user experiences. By adding the login module, the parents can track and manage their child's

activities within the app and make sure they log in securely.

ii. Image to Story Generation module

The image to story generation module enables generating a personalized story based on the user's ideas that process visual inputs, either from in-app drawing modules or uploaded photos from gallery. The children can choose a story genre such as adventure, mystery, or fairy tale before generation to guide the story tone. After uploading a photo or drawing, the application will generate a story draft based on ideas uploaded. The children are allowed to review and edit the generated story. Image to story generation module provides a way for children to connect with visuals.

iii. Drawing module

The drawing module allows children to create visual content directly within the application. The children can draw their own ideas based on their imagination creatively instead of uploading an image from a gallery that limits their ability of drawing. They can also choose to save their drawings without uploading them to generate a story. The drawing module includes features such as providing a user-friendly drawing space equipped with brushes, color palettes, and shape templates. The drawing module encourages imaginative thinking through freehand drawing and visual storytelling.

iv. Community Sharing module

The community sharing and interaction module enables users to share their generated stories and drawings through a shared community space to the other users, which introduces a social element to the application. In the community module, users can explore the stories created by others and express their appreciation through likes, comments, and shares. The users can also provide their feedback through a safe community comment module. The interactive community module design will be safe for children in mind by providing a filtered and positive comment for them to prevent inappropriate language. Other than that, users can also share the post within community to other users through the user chat sharing module. The module enables children to receive encouragement from other users and explore different kinds of storytelling ideas, which can enhance their creativity and imagination.

v. History Tracking module

The story history tracking module saves every record of a story created by users. After each time a child completes their story, the story will be auto saved to their personal profile to let them track their previously generated stories, view, and edit the stories later. The module not only allows users to track back their generated stories but also keeps track of their growth over time, which supports long term creative development. The parents and educators can review their child's learning process and engagement with the application by using the history tracking module.

vi. Text to Speech module (TTS)

Text-to-speech module enables users to convert the written text-based stories to spoken audio, which increases the accessibility of the application. TTS features are especially important for children who have not yet fully developed reading skills and who benefit from auditory learning. The TTS module provides an adjustable playback speed to address user's different kinds of favorites and needs. The module can reinforce vocabulary, enhance comprehension, and add a dynamic element to the storytelling experience by allowing the children to listen to their own stories read aloud.

vii. Story Editing module

The story editing module enables users to review and edit their generated story from drawing or uploading photos. For instance, children can adjust the descriptions, modify characters, and extend the storyline based on their ideas and imagination. The text editor is designed with large fonts and simple controls to suit younger users.

viii. Story Management module

The story management module handles all story data operations. It manages story versioning switching, allowing users to switch between original and enhanced versions. The module provides efficient story retrieval with caching per performance, support filtering and searching operations for users. This will ensure data persistence across app sessions. The users are able to store their generated stories, and with the history tracking module for progress updates.

ix. Story Searching module

The story searching module allows the users to search for stories using full-text search across title, author, and categories. It provides real-time search as users type, with case-insensitive and partial word matching. The module highlights matching terms in search results and maintains search history for quick access to previous queries. The search interface is easily accessible from the library screen and displays results in a familiar story card format.

x. Story Filter Module

The story filter module enables the users to use the filtering capabilities to organize and view their story based on various criteria. It includes filters for All stories, My Drawings, Generated Stories, Continue Reading, Saved (Community), and Favorites. The module implements dynamic filter state management with immediate UI updates when filters are applied. Filter state can be persisted across sessions, and the active filter is clearly displayed in the interface.

xi. User Profile Management module

The user profile management module enables users to do profile customization and account management. It allows users to set and edit custom unique usernames with validation and manage profile pictures by selecting from app avatars or importing from device gallery. New users automatically receive default avatars generated from their email's first letter. The module displays comprehensive reading statistics, such as total reading time, favorites count, drawings saved, and stories created.

xii. Story Export and Sharing module

The story export and sharing module enables users to export and share content outside the application. It provides formatted story text export including title, description, author, categories, age range, reading time, and complete story content with all pages. Exported text can be copied to clipboard with one-tap functionality. The module enables saving story cover images and drawings to the device gallery using native platform integration. It supports exporting both original and enhanced drawings versions.

xiii. Settings and Preferences module

The setting and preferences module manages application wide user preferences and configuration. It stores TTS preferences for voice selection and speech rate, and reading preferences for text size, night mode, auto-scroll, and highlights.

xiv. User Chat Sharing module

The user chat sharing module is an in-app friend chat sharing feature. Users can share their story or post previews directly in one-to-one chat conversations, which enabling safer peer-to-peer content sharing inside the app ecosystem. They can also click into the public profile to see all posts posted by that user within the community.

xvi. Simplified Dictionary Checking module

This module allows users to tap or select difficult words and view simplified, child-friendly meanings to support vocabulary learning and reading comprehension during storytelling and story reading.

1.5 Proposed Approach / Study

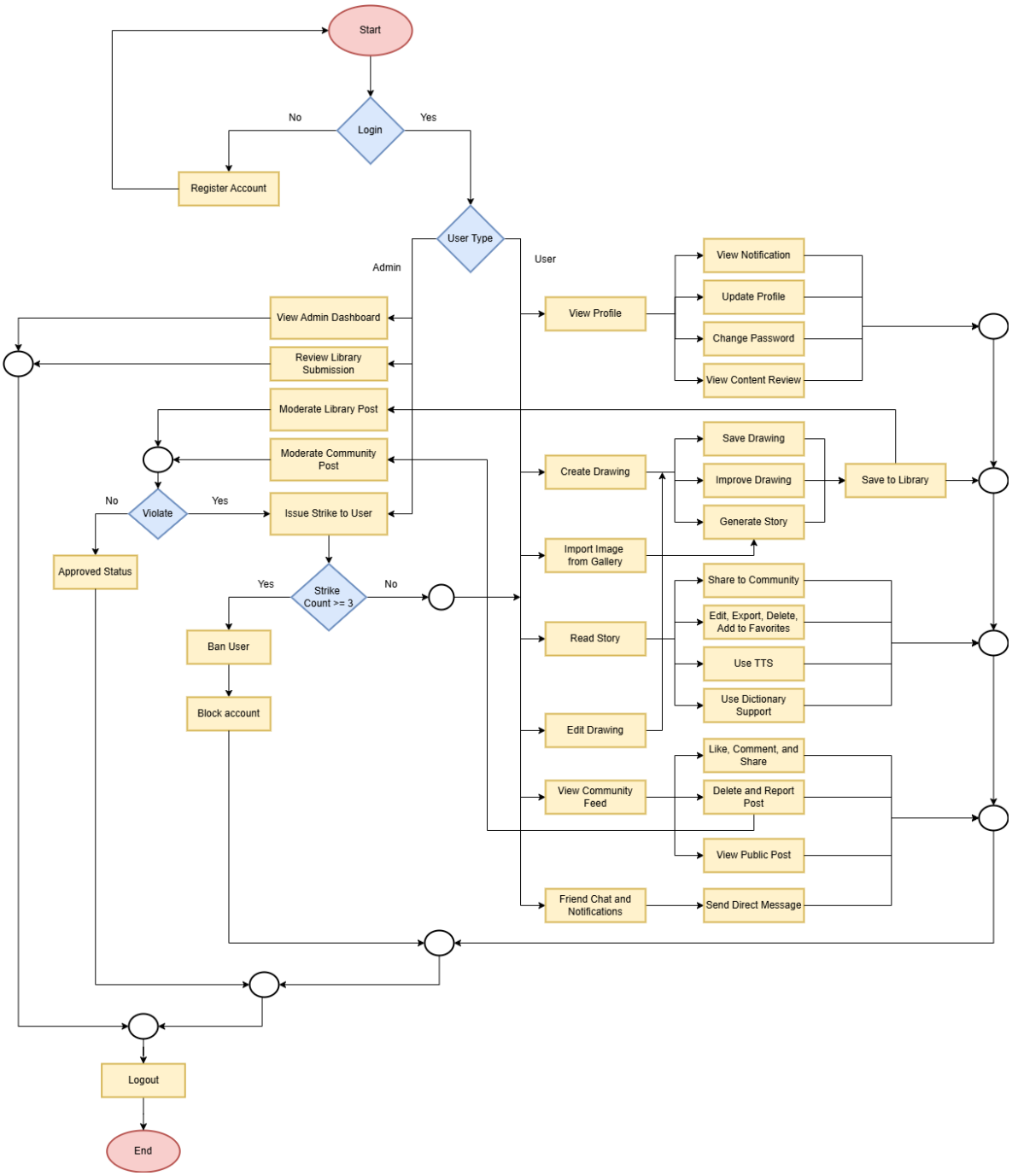


Figure 1.5.1 Flowchart of Child-Centric Interactive Storytelling Application

Figure 1.5.1 illustrates the overall system flowchart for the StoryCraft application, which separates into two primary roles after successful login, such as administrative and user. Regular users are allowed access to drawing creation, AI story generation, interaction reading tools including TTS and dictionary support for story readers, community feed engagement, profile viewing, editing story, and friend chat. While admin are allowed to access to a moderation

dashboard where they review submissions and perform platform safety approving. If content violations are detected, the system executes a disciplinary flow that issues strike and automatically blocks the user's account upon reaching 3 strikes limit.

1.6 Highlight of What Have Been Achieved

The project has successfully implemented a working prototype of StoryCraft with end-to-end storytelling flow. The system supports visual-to-story creation by allowing users to generate stories from both in-app drawings and imported images, with genre-based outputs and editable story text to encourage personalization. To support young learners, the application includes child-friendly creative tools, such as an intuitive drawing canvas, enhancement support, and simplified user interaction design.

In addition, the project implemented a moderation-centered safety architecture that includes image moderation, story-text moderation, and clear pending or rejected review states to ensure safer content handling. Core library and story lifecycle capabilities are also completed, including save, retrieve, edit, search, filter, version handling, and history tracking. Beyond individual creation, Story Craft integrates social functionality through moderated community sharing, likes or comments, and peer-to-peer sharing via in-app chat.

From an accessibility and usability perspective, the application includes text-to-speech (TTS), export and copy features, simplified dictionary checking, user profile management, and configurable preferences to improve overall user experience. Technically, the project was developed using Flutter and validated through Android-focused workflow testing, demonstrating a maintainable architecture with practical potential for future extension.

1.7 Impact, Significance, and Contribution

The StoryCraft project demonstrates practical impact on children's digital storytelling engagement, creative expression, and interactive learning. By integrating drawing, image-to-story generation, editing, moderation, and community interaction in one mobile platform, the project addresses gaps found in many existing children's storytelling apps that rely heavily on fixed templates and passive reading experiences.

A key significance of this project is that it shifts children from content consumers to content creators. Instead of only reading predefined stories, children can draw or upload images, generate stories from their own visual ideas, select preferred genres, and edit generated text. This creator-centered approach supports imagination, self-expression, storytelling confidence, and early literacy development through active participation.

The project also contributes a child-friendly creation workflow through simplified drawing interactions and visual support features, reducing barriers for younger users with varying skill levels. By connecting visual input directly to narrative output, StoryCraft strengthens the relationship between creative art activity and language-building outcomes, which is valuable in early childhood learning contexts.

Another major contribution is the implementation of a moderated social storytelling environment. Children can share approved stories, view peers' creations, and interact through controlled community features such as likes, comments, and friend sharing. This adds collaborative motivation and social learning opportunities while maintaining safety through moderation and review controls.

From a technical perspective, the project contributes an end-to-end Flutter-based mobile solution that combines AI-assisted storytelling, moderation-aware content lifecycle management, text-to-speech accessibility, profile and personalization, and library management features in a single architecture. This provides a useful reference model for future child-centric educational application development that balances creativity, usability, and safety.

1.8 Report Organization

This report is separated into 7 chapters, and the details of each chapter are shown as following:

Chapter 1 provides the background of the StoryCraft application. It outlines the problems statement regarding current storytelling application, shows the primary project objectives and scopes, what have been archived, impact, significance, and contribution and introduces the overall organization of the report.

Chapter 2 explores the existing literature review relevant to the project. It reviews existing applications in the market. The strength and weakness of each existing application is also listed and compared in the table.

Chapter 3 shows the software development methodology used to develop the application. It also outlines the project timeline, hardware and software requirements,

Chapter 4 presents the use case diagram for StoryCraft application, use case description for each use case. Other than that, the system development of the code including frontend and backed development is also explained.

Chapter 5 provides the final development application for implementation and testing. It represents a comprehensive walkthrough of the user interface and user experience design concepts.

Chapter 6 analyzes the final outcome of the application. It provides performance metrics and evaluation against project objectives in Chapter 1 and discuss the challenges during development.

Chapter 7 summarizes the entire project journey and highlights the novelties and contributions of StoryCraft. Besides, this chapter also mentions the future works for enhancement

CHAPTER 2 LITERATURE REVIEW

2.1 Overview

This chapter will discuss and review existing applications and projects that are relevant to storytelling. A few examples of existing applications are used as guidance while developing a new storytelling application. In addition, a review of the existing project is to understand the project objective and the process of developing the project. The existing applications are reviewed, and their strengths and limitations are compared to providing a summary.

2.2 Existing Applications

2.2.1 Readmio

Readmio is a mobile storytelling application that is especially designed for children and families. Readmio believes that through storytelling, kids can become smarter and kinder and families happier [3]. The app provides a chosen collection of stories enhanced with interactive features and sound effects to promote early literacy development and parent-child interaction. The main purpose of Readmio is to transform traditional reading stories into an engaging audio-visual experience [7]. Readmio does not support and serves as a customizable or generative story platform but functions as a guide for user interface design and content organization in child-centric storytelling. Readmio enhances storytelling by synchronizing sound effects with a parent's live reading. Also, it focuses on audio experiences to reduce screen time, offers offline text-to-speech playback, and audiobook creation tools [3]. The app supports both iOS and Android platforms [7]. After downloading the Readmio application through the Play Store on the smartphone, it will first prompt out a short brief description about the app. By continuing, Readmio requires users to login through Google or Apple accounts as shown in Figure 2.2.1.1. But the user also can choose to continue without an account as a guest. Readmio allows users to choose language for reading as shown in Figure 2.2.1.2 Language Selection. In addition, Readmio organizes its story collection by age group as shown in Figure 2.2.1.3 Browse Categories. For instance, age 3+ put in one category, age 5+ put in one category, age 8+ put in one category, and bedtime stories put in one category. In each category, it shows related stories and details for themes such as family, friendship or adventure and the estimated reading time of each story as shown in Figure 2.2.1.4. The app shows an audio label on the right bottom of each story.

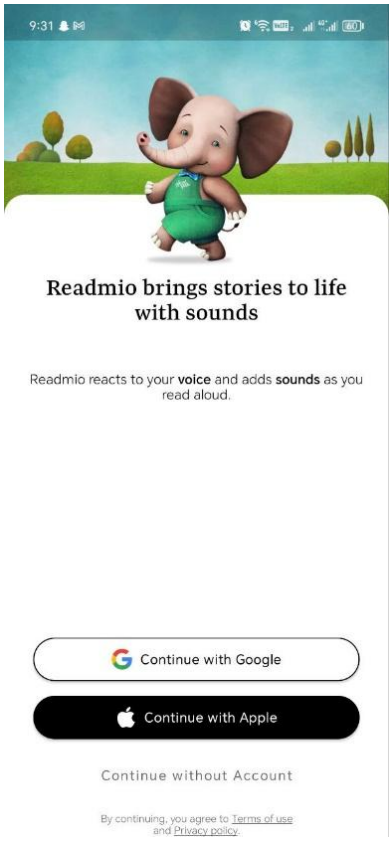


Figure 2.2.1.1 Login Page

Figure 2.2.1.2 Language Selection

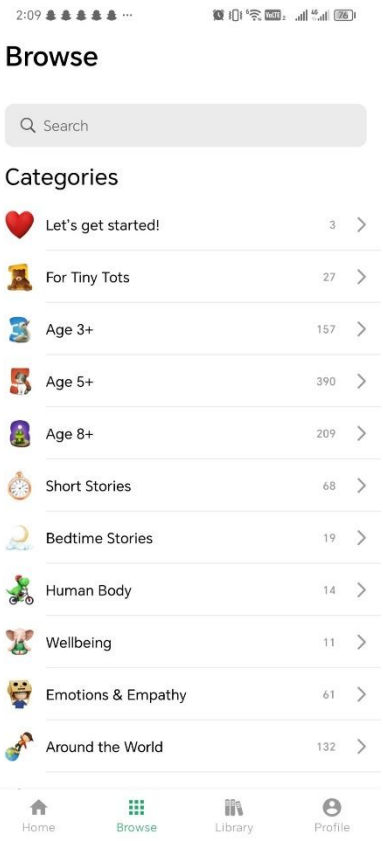


Figure 2.2.1.3 Browse Categories

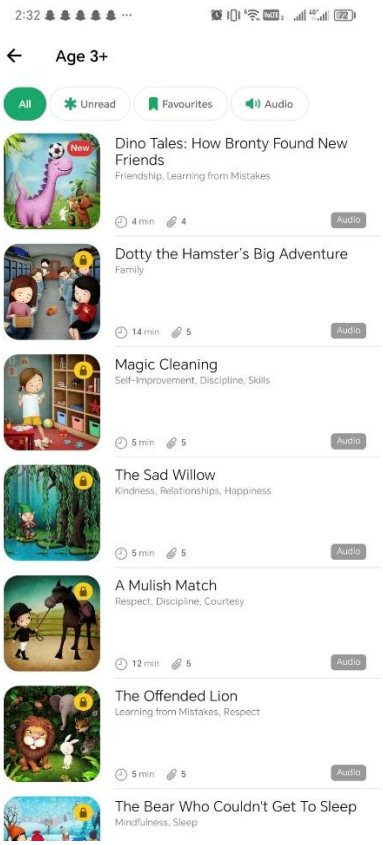


Figure 2.2.1.4 Age 3+ Category

Furthermore, Readmio provides a library management system for users who can access a personal library that stores previously opened stories as shown in Figure 2.2.1.5. Users do not have to worry about not being able to find the stories they have browsed. These stored stories can be sorted into different categories in order to find out their preference's story in a faster and easier way. The categories included "All," "Unread," "Favorites,", "Audio", and "Recording". However, most of the stories require a paid subscription to unlock the story as shown in Figure 2.2.1.6, only a limited number of stories are accessible in the free version. For instance, Readmio only offers a free story daily and several stories on trial. For the premium content lock behind a subscription model version, the users only can preview the story for the first two pages. This limits the usability for users who cannot afford the premium tier.

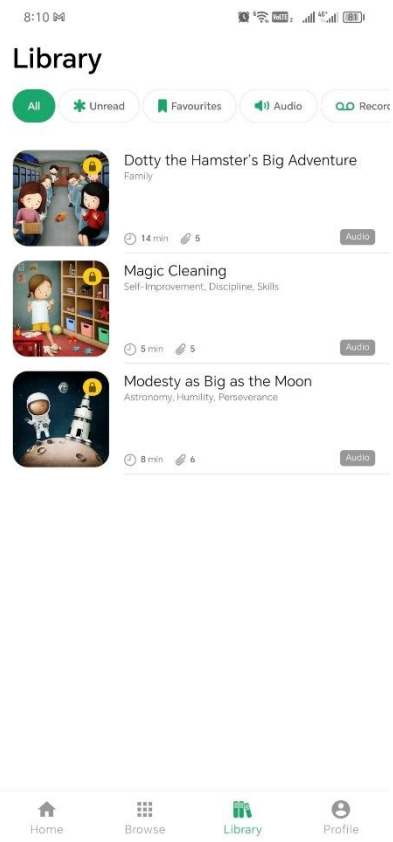


Figure 2.2.1.5 Library Management

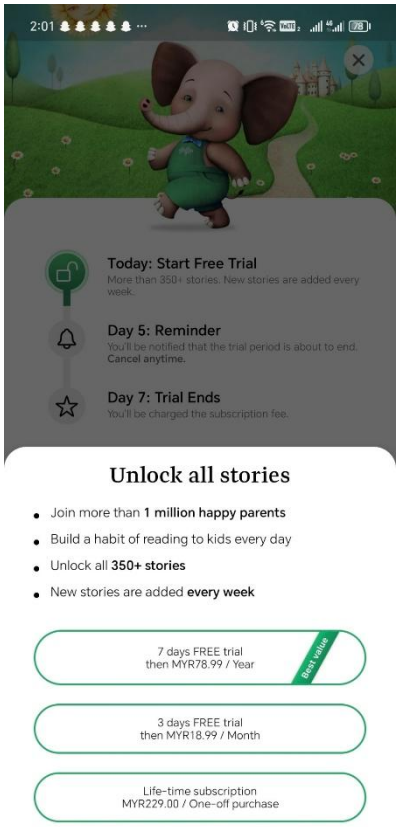


Figure 2.2.1.6 Paid Subscription

Readmio supports text-to-speech and voice narration features for users to focus on listening, especially for those who have reading difficulties on words and younger children. Most of the stories in the Readmio app are available in audio format as shown in Figure 2.2.1.5. The text-to-voice feature makes it accessible for pre-readers and children with visual impairments. The users can choose to read in text form or listen as sleep music as shown in Figure 2.2.1.7. If the

user chooses to read in text, they can set the brightness, night mode, and size of the text shown in the story according to their preferences as shown in Figure 2.2.1.8. Other than that, the user also can record their sound recordings by clicking the record button at the top middle as shown in Figure 2.2.1.8 and save it to the library. To find out their recording, users can get it from the library recording section as shown in Figure 2.2.1.5. The voice narration is smooth and child-friendly, and it is an option for parents to read the story aloud while background sounds enhance the atmosphere by setting the time wanted to be played either in 30 minutes, 1 hour, 3 hours or nonstop.

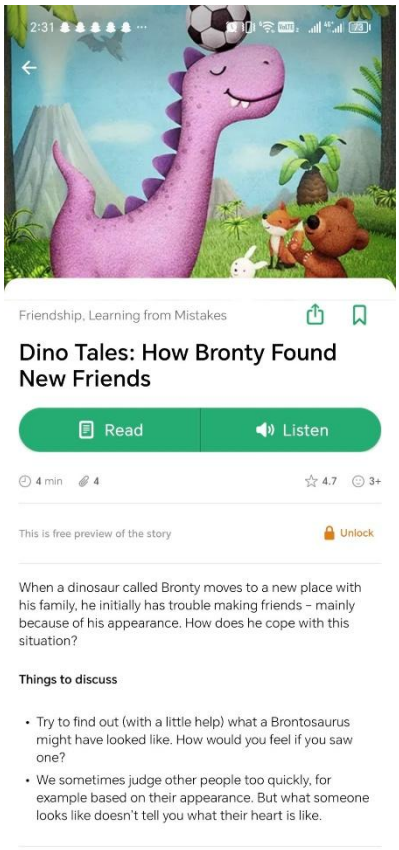


Figure 2.2.1.7 Read or Listen Choice

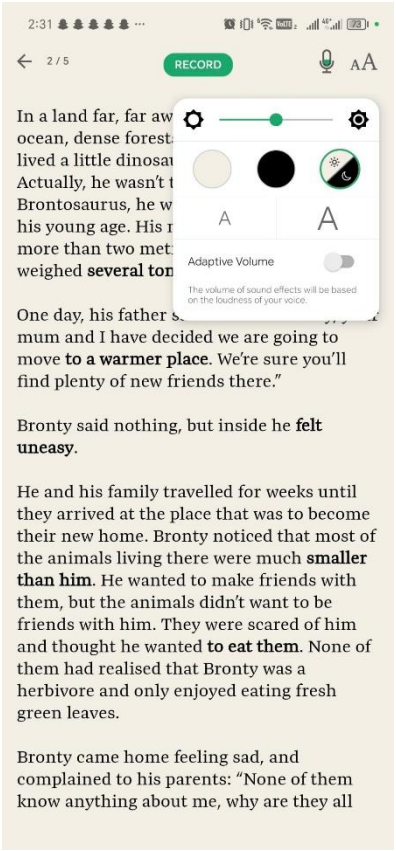


Figure 2.2.1.8 Read Text Setting

Strengths

- Well categorized story library sorted by age, theme, and reading time.
- Simple navigation with large icons and colorful visuals designed.
- Audio enhanced by building in narration with sound effects.
- Support audio experiences, offline text-to-speech playback, and audiobook creation tools.
- Recording parent narration and voice narration features make it engaged.

Weaknesses

- No customization features like drawing and story creation.
- Children cannot create and personalize stories without paying.
- No community interaction like sharing their stories, likes, and comments.
- Most stories are pre-written and locked behind subscription plans.

2.2.2 Short Stories

Short Stories is an educational mobile reading platform application developed by Eduteco [12]. Short Stories is designed to promote independent reading among children aged 5 years and above. Based on pedagogical and psycholinguistic these two principles, the use of short stories in language learning aims to develop reading comprehension, vocabulary acquisition, and pronunciation skills by providing contextualized input in an interactive and friendly environment [4]-[7]. The app is designed with simple UI elements to enhance literacy development through interactive reading experiences.

Short Stories app provides a virtual library of children's literature that focuses on a wide variety of well-known classic fairy stories and fables. These stories are carefully selected to draw the attention of children and foster their interest in reading. Examples of stories included The Three Little Pigs, Cinderella, The Ugly Duckling, and Little Red Riding Hood. Stories are grouped into different themes such as Classic Tales, Princesses, Fables, and Adventures. Each story book is visually printed with a realistic interactive book cover and shelf-style one page layout which is clear and simple for children to find out their preference story books as shown in Figure 2.2.2.1. Besides, each story book contains only a maximum of 30 pages. While each page contains very short simple texts at top and a color animation photo which is related to the text given at the bottom of page as shown in Figure 2.2.2.2. These kinds of designs help children to practice their reading independently and further improve their imagination while reading when they link each page of text with related photos. The Short Stories app provides a set of navigation tools to enhance reading experience, such as forward and backward arrows buttons allowing page by page reading. There is also a “Return to Start” button designed for users to enable restarting the story instantly. A progress checker like “6/22” placed between two forward and backward arrows as shown in Figure 2.2.2.2 helps the reader to keep track of their current read position. Story Stories provides a read-aloud feature that allows readers to listen to the text of each current page with a natural voice. In addition, the app also provides a

slowdown in pronunciation of words for readers. For instance, the reader can tap on individual words to hear them pronounced more slowly to capture each sound distinctly one by one.

Figure 2.2.2.1 Shelf-style Main Page

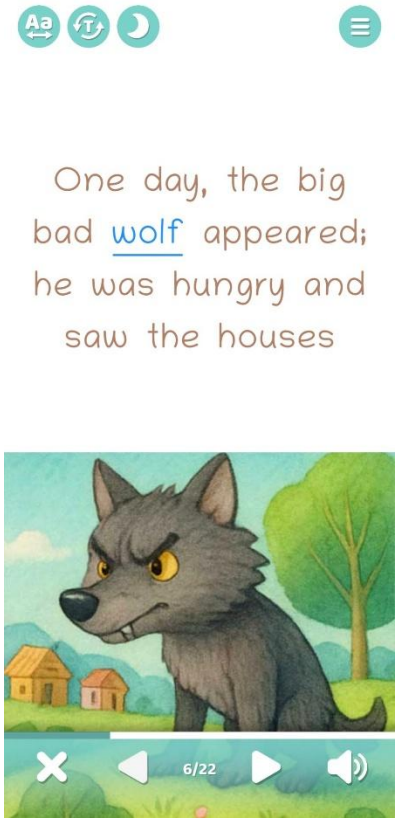


Figure 2.2.2.2 Short Text with Photo

Furthermore, the Short Stories app provides up to 4 different types of fonts for users to choose from and text cases like all capital letters or mixed capitals depending on their preferences. The reader can switch the mode between night and day to prevent the damage caused by continuous screen exposure and promise comfortable reading. All these feature icons can be found in Figure 2.2.2.2 at the top left side. Besides, the app also supports different languages switching between Spanish, German, Italian, and Portuguese as shown in Figure 2.2.2.3. Short Stories app provides a security check feature as shown in Figure 2.2.2.4 like calculating a simple addition calculation if the users want to change the setting inside the app. The security check feature can promise the parents by providing a safety space for their child when using the app. The user needs to unlock a story with a free key provided. When the children want to unlock more books, they need to buy a paid subscription to unlock additional keys. To prevent them from making the decision without their parent's permission, they need to answer an additional calculation first for a security check which is harder in their current learning level to proceed to the next payment page.

Figure 2.2.2.3 Language Switching Setting

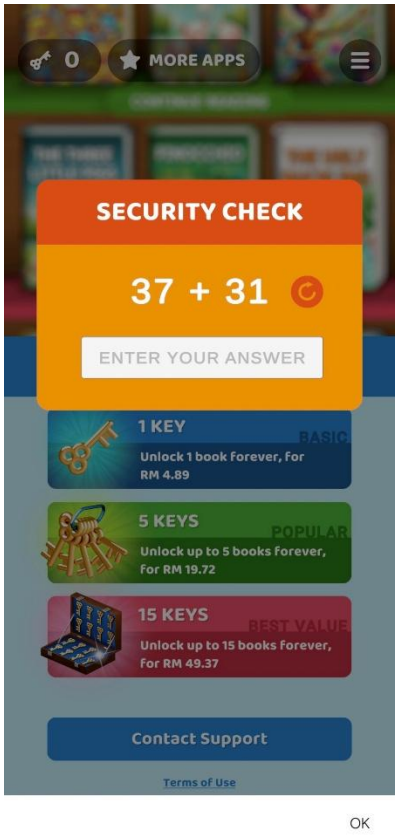


Figure 2.2.2.4 Security Check

Strengths

- Support multiple languages, fonts, text cases, and night and day mode.
- Stories are shown in simple short text with related photos.
- Read-aloud option and slowdown word pronunciation features supported.
- Shelf-style main page with interactive book covers for easy browsing.
- The security check feature adds parental control for settings and purchases.

Weaknesses

- No drawing and story customization features.
- Lacks offline listening support.
- No community interaction for children to share or engage socially.
- Most stories are pre-written and locked behind subscription plans.

2.2.3 ReadToMe

One of the existing applications relevant to the project is ReadToMe, which is a mobile application designed to provide literacy and reading engagement among children [13]. ReadToMe provides a digital library of books categorized by age group, reading level, grade level, book type, and subject category for users to select their preference reading materials as shown in Figure 2.2.3.1. It is easier for both parents and children to choose the books faster and select depending on their child's current studying level. From the age group category, it caters to different age ranges from toddlers to early primary school students and covers a wide range of topics such as colors, science, animals, and character education. Furthermore, ReadToMe provides two types of reading mode, the first one is just listening without any provided text, and the second one is listening with provided text and interactive color photo in each page as shown in Figure 2.2.3.2. For just listening without provided text, it will show a headphone symbol in the cover page of the storybook.

Figure 2.2.3.1 Categorized Library

Figure 2.2.3.2 Read in Text or Listen Only

For the listening with provided text and interactive color photo, the app provides dual reading modes, the "Read To Me" option narrates stories aloud to assist early learners, and the "Read Myself" option supports independent reading as shown in Figure 2.2.3.3. The story will

automatically switch pages read aloud after one page is done and continue to read until the end of story. Other than that, there is also a slow narration function that allows users to slow down the reading speed based on their level for different learners. The slow speed button is put in the top left corner as shown in Figure 2.2.3.4. The app allows readers to turn the page by clicking the big and simple arrow buttons or stopping the story. The app provides a splash card for readers to preview their selected story or jump to the pages they want as shown in Figure 2.2.3.5.

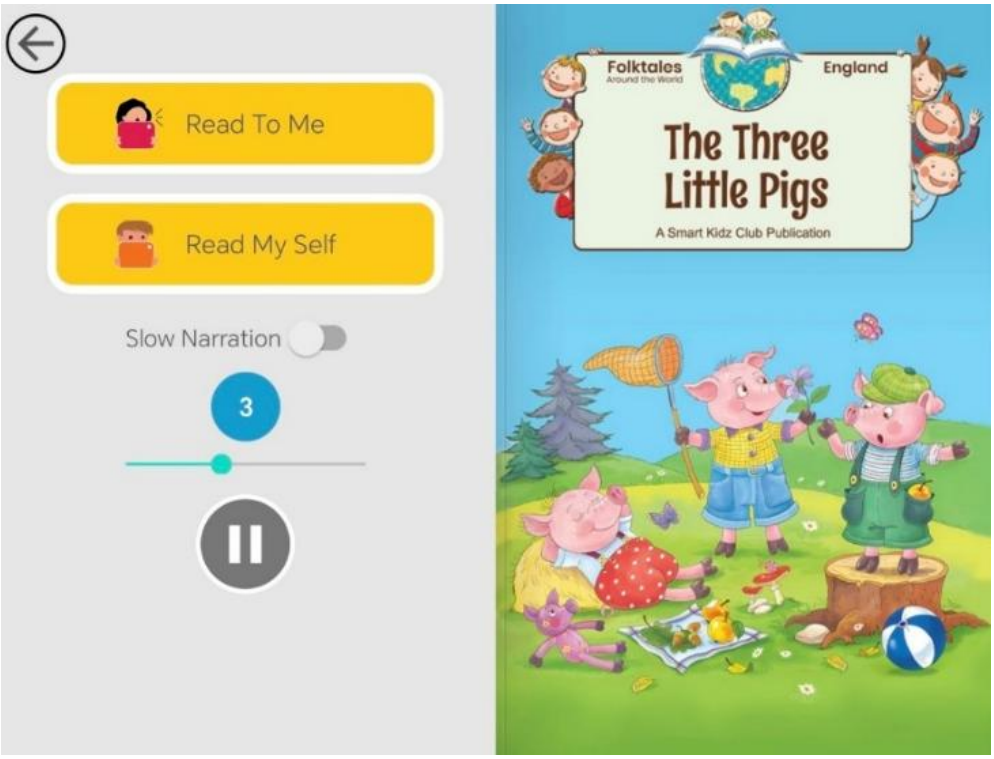


Figure 2.2.3.3 Two Reading Mode

Figure 2.2.3.4 Slow and Switch Page Button

Figure 2.2.3.5 Flash Card Preview

Besides, the ReadToMe app provides a search and filter book feature for readers to search their wanted story book based on their requirements by grade, reading level, age group, or category as shown in Figure 2.2.3.6. The offline reading feature enables children to continue learning even without internet connectivity. The reader can find their preferred filtered story book faster and easier. The app provides an interaction with big icons and cartoons to represent each multimedia learning resources trending that can search through the library such as flashcards, audiobooks, and hearables as shown in Figure 2.2.3.7.

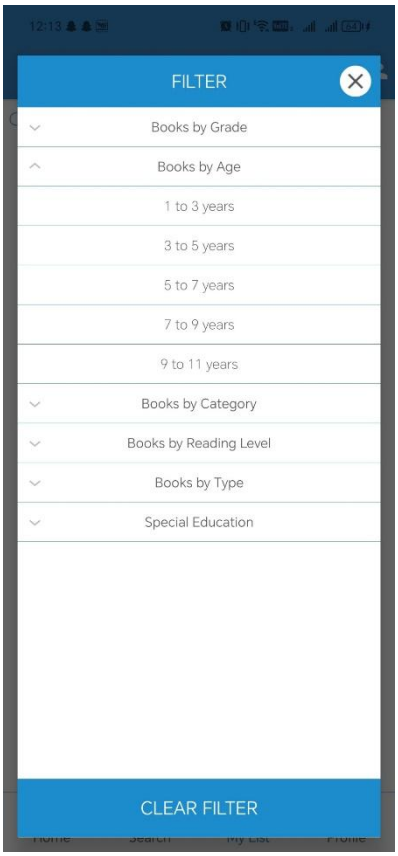


Figure 2.2.3.6 Filter Book Feature

Figure 2.2.3.7 Search Engine Library

Strengths

- Books are organized by age, grade, reading level, and subject.
- Dual Reading Modes: “ReadToMe” narration aids children with limited reading skills, while “Read Myself” promotes independent reading.
- Offline reading is supported.
- Dedicated section for basics, colors, shapes, and social stories caters to diverse learning needs.
- Visual and interactive design with big icons, flashcards, and audiobooks.

Weaknesses

- Users cannot select or change genres for storytelling.
- Children cannot create stories from their own drawings or visuals.
- Stories remain pre-written and subscription based.

2.3 SUMMARY

In this chapter, three storytelling applications like Readmio, Short Stories, and ReadToMe have been reviewed. Readmio focuses on creating an immersive family reading experience by synchronizing narration with sound effects, supporting text-to-speech, and allowing parent voice recording, though most of its content is locked behind subscription plans and it lacks customization or community features. Short Stories emphasizes language learning and independent reading with multilingual support, different fonts, and a slowdown pronunciation feature; however, it does not support offline listening and requires subscriptions for most stories. ReadToMe provides a structured digital library with dual reading modes, slow narration, and offline access, but it does not allow genre customization or story creation. Overall, the three applications contribute to literacy development in different ways: Readmio excels in interactive audio storytelling, Short Stories in language acquisition and accessibility, and ReadToMe in flexible structured reading. Their common weaknesses are the lack of story customization, community interaction, and dependence on subscription-based models.

Three existing mobile applications are reviewed which are Readmio, Short Stories, and ReadToMe. The strengths and weaknesses are compared and summarized in Table 2.3.1.

Table 2.3.1 Comparison of Strengths and Weaknesses of Existing Application

	Readmio	**Short Stories**	**ReadToMe**
Target Audience	-Children age 3+; focus on family storytelling.	-Children age 5+; focus on independent reading and language learning.	-Toddlers to primary school students; focus on literacy engagement.
Content Type	-Interactive stories with audio effects; categorized by age, theme, time.	-Classic fairy tales and fables like Cinderella, Three Little Pigs, and others; short text and images.	-Wide range of books: categorized by age, grade, level, subject; includes educational topics.
Text-to-Speech (TTS)	-Full TTS, narration sync with sound effects, offline support, parent recording option.	-Read-aloud per page, slowdown word pronunciation and no offline TTS.	-Dual modes: "*Read To Me" for* narration and *"Read Myself"*, slow narration option.
Genre Choice	-No custom genre selection; library pre-set by age and theme.	-No custom genre choice, stories grouped by theme only.	-No user-controlled genre customization, only have pre-organized by app.
Reading Modes	-Text reading, text-to-speech, audiobooks, and parent voice recording.	-Independent reading, read-aloud, and slow pronunciation.	-Narration (*Read To Me*) or independent (*Read Myself*), slow narration.

Accessibility	-Adjustable brightness, night mode, text size. -Text-to-speech for visually impaired.	-Multiple fonts & text cases. -Language switching between Spanish, German, Italian, Portuguese. - Night and day modes	-Large icons, simple UI. -Filter by age, grade, subject for tailored learning.
Offline Support	-Supports offline text-to-speech and audiobook playback.	-No offline listening support.	-Offline reading supported.
Interactivity	-Syncs sound effects with reading. -Allows voice recording by parents. -Child-friendly narration.	-Interactive book covers in shelf style. -Short text and image association. -Security check for parental control.	-Interactive flashcards. -Filter and search tools. -Navigation with big icons.
Customization	-No drawing and story creation, only with subscription.	-No customization or drawing features.	-No story creation or customization.
Community Features	-No sharing, liking, or comments.	-No sharing, liking, or comments.	-No sharing, liking, or comments.
Unique Strength	-Immersive audio storytelling with sound effects and narration sync.	-Language learning support with slow word pronunciation and multilingual options.	-Most flexible reading experience because have dual modes, offline, filtering by grade and subject.

CHAPTER 3 METHODOLOGY AND TOOLS

3.1 Overview

In this chapter, the methodology and technology used to develop Story Craft mobile application is explained here. Hardware requirements, software requirements, and the steps of methodology will be explained. The project timeline has also been planned to ensure the project is completed on time.

3.2 Methodologies

The methodology used for this project was Agile development as shown in Figure 3.2.1 and Figure 3.3.2. Agile is an iterative and incremental process approach that allows continuous feedback and improvements at each stage. The methodology breaks down the project process into small and manageable units called sprints. Agile was selected for this project because the app involves multiple interconnected modules that required continuous refinement, especially when integrating AI-generated content, moderation logic, and social features into one mobile application. The development process was carried out iteratively in short cycle so that each cycle produced a working increment that could be tested and improved before moving to the next phase. In FYP2, Agile was used to extend the FYP1 baseline with moderation, communication safety, chat sharing, and performance refinements. Each sprint ended with integration testing because Firebase and AI services required repeated rule and API alignment.

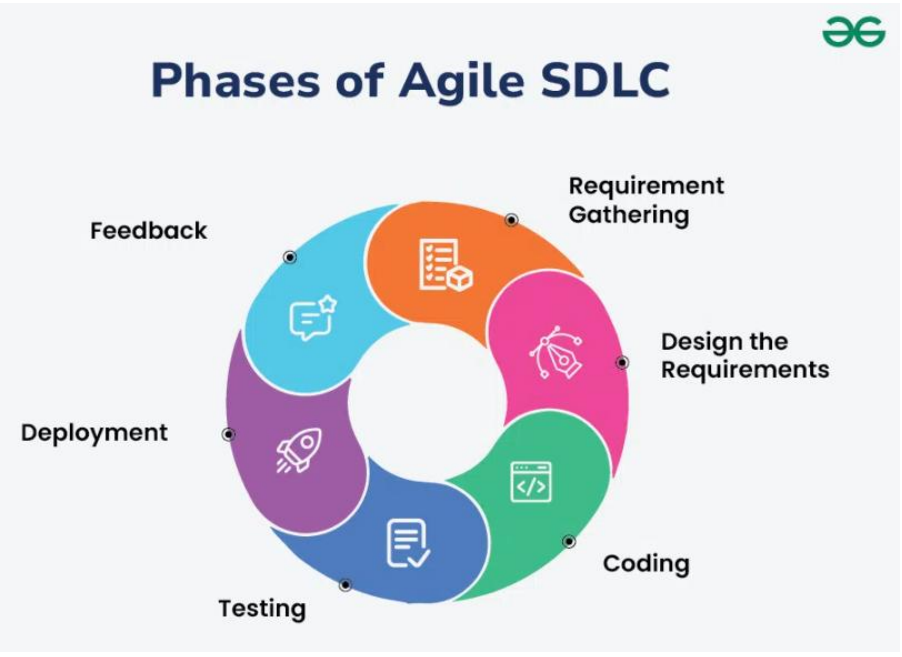


Figure 3.2.1 Phases of Agile Methodology SDLC [10]

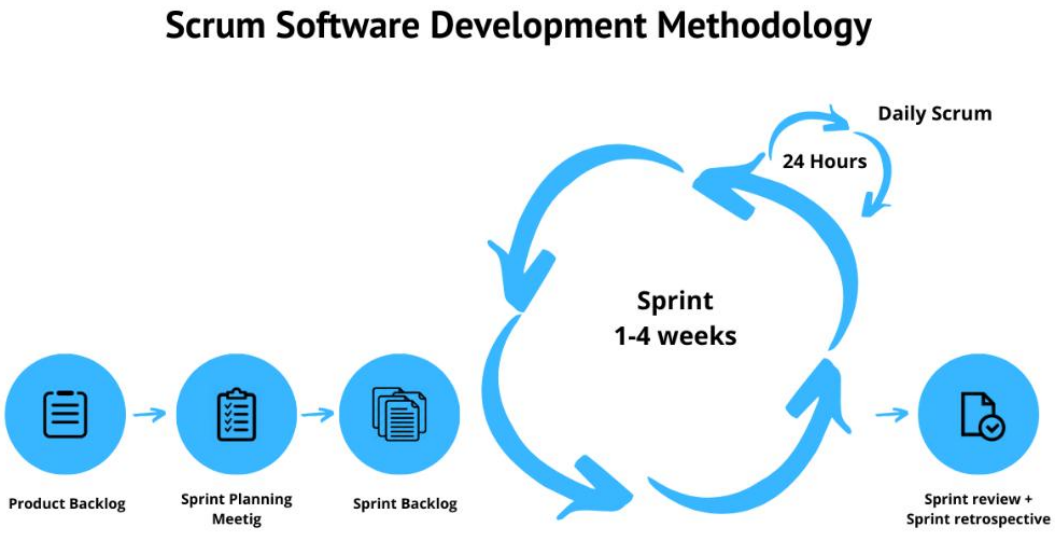


Figure 3.2.2 Phases of Agile Methodology SDLC [11]

3.2.1 Product Backlog Creation Phase

The first step of the agile development methodology was creating a Product Backlog. Product Backlog is a list of tasks that are required to complete the Story Craft app. In FYP2, the backlog was updated from FYP1 baseline and reorganized based on implementation priority, integration risks, and user value. User stories remained the main way to define requirements from the user perspective, particularly for children and supervised usage contexts. The examples of user stories included generating narratives from drawings, reading with text-to-speech support, sharing content safely in the community, and interacting through friend chat were translated into technical tasks for UI development, backend integration, and moderation handling. Throughout development, backlog items were continuously reprioritized when new constraints appeared, such as firestore permission dependencies, moderation outcomes, and performance behavior during image-heavy flows.

3.2.2 Sprint Planning Phase

Sprint Planning is the phase that selects a set of high priority tasks from the previous product backlog to be completed within a period, 1-4 weeks per sprint and group them into time-boxed implementation periods. For FYP2, planning was aligned with the semester timeline until report completion, followed by final documentation and presentation preparation. Each sprint focused on a specific delivery objective, including community sharing, safe and moderation, social feature extensions, reader and literacy support, performance and resilience, final

integration and testing, and documentation. This structure ensured that every cycle delivered a testable increment rather than isolated partial components, and it helped maintain progress visibility for supervisor review and report tracking. For this project, the sprint planning for FYP1 and FYP2 included the following breakdown as shown in Table 3.2.2.1 and Table 3.2.2.2.

Table 3.2.2.1 Incremental and Sprint-based Plan for FYP1

Sprint	Goal	Deliverables
1	Project setup and authentication	Configured Flutter and Firebase, implemented login and registration, and completed basic user profile management.
2	Drawing module	Developed a custom drawing canvas with brush tools, color picker, undo and redo, and save functionality.
3	Story generation from drawing and image upload	Built genre selection, integrated AI generation API, generated stories from drawing/image input, and stored generated outputs.
4	Story reading and editing	Implemented story reader with page navigation and reading progress tracking, and completed story text editing with page/content management.
5	TTS, discovery, profile, and settings	Integrated text-to-speech like voice and speed control, implemented story discovery like search, filter, and favorites, and completed profile and settings features.
6	Final integration and testing	Performed bug fixing, UI consistency improvements, performance tuning, and end-to-end module testing for FYP1 submission readiness.

Table 3.2.2.2 Incremental and Sprint-based Plan for FYP2

Sprint	Goal	Deliverables
1	Community sharing module	Community tabs, posting workflow, likes, comments, and share, notification handling, story-to-community submission from story detail.
2	Safety and moderation	Library moderation, community moderation, image moderation and text moderation integration, comment moderation, content review screens.
3	Social feature extensions	Friend list and inbox, chat UI, shared post preview cards in chat, community friend chat notifications.
4	Reader and literacy support	Simplified dictionary and child-friendly meaning support.
5	Performance and UX hardening	Firebase and Firestore rule alignment with features, permission denied handling improvements, import image performance, and loading UX refinements.
6	Final integration and testing	End-to-end passes across auth, drawing, generation, library, reader, community, chat, moderation.
7	Documentation and submission	Report content, diagrams, prepare screenshots, demo preparation, final polishing, submission milestones.

3.2.3 Design and Prototyping Phase

The design and prototyping phase focused on transforming backlog requirements into practical screen flows before implementation. Interface decisions prioritized child-friendly interaction, including clear navigation, readable layouts, and guided actions for important tasks such as drawing, generating stories, reading, and sharing. Navigation flow was designed around the application's main tabs, and transitions between screens were refined so users could move logically from creation to reading and then to sharing. In FYP2, additional design attention was

given to safety-related interactions, including moderation-aware actions and explanatory dialogs, so users receive clear feedback when content is pending review, approved, or restricted.

3.2.4 Implementation and Coding Phase

Implementation was carried out using Flutter and Dart for the mobile front end, while Firebase services were used for backend capabilities. Firebase Authentication handled user access, and Cloud Firestore and Storage supported story data, profile information, community content, and related social interactions. AI-assisted features were implemented through integrated services for image-to-story generation and content moderation support, while additional modules such as friend chat, community notifications, dictionary assistance, and update or version handling were developed and integrated into the same application architecture. During this phase, coding work also included service-layer organization, UI-state handling, and cross-module integration so that the final app behaviour matched real user flows.

3.2.5 Testing and Review Phase

Testing was conducted after each major implementation cycle to verify functionality, usability, and integration stability. In FYP2, testing emphasized practical module validation across real feature flows, including authentication, drawing and image input, story generation, reading and editing, community posting, moderation outcomes, and chat sharing behaviour. UI responsiveness and interaction quality were reviewed across typical device usage conditions, while integration testing focused on service connectivity, data consistency, and error handling between modules. Review sessions with the supervisor were used to confirm progress, identify defects, and determine priorities for subsequent improvements before final submission.

3.2.6 Retrospective and Continuous Improvement Phase

At the end of each cycle, retrospective evaluation was used to reflect on completed work, identify technical blockers, and define improvements for the next iteration. This continuous improvement process was important in FYP2 because several features required repeated tuning after integration, particularly moderation flows, user feedback messaging, and overall performance in image-related operations. Feedback from testing and review activities was translated into concrete revisions such as improving loading and status communication,

refining validation behaviour, and strengthening reliability across connected modules. Through this iterative process, the project moved from feature-level completion toward a more stable and coherent final system suitable for FYP2 assessment.

3.3 Tools to Use

i. Hardware Requirements

Table 3.3.1 Hardware Computer for Mobile Application Development

Component	**Requirements**
Model	OMEN by HP Laptop 15-dc0xxx
Operating System	Microsoft Windows 11 Home Single Language
Processor	Intel(R) Core(TM) i7-8750H CPU @ 2.20GHz (2.21 GHz)
RAM	24.0 GB (23.8 GB usable)
Storage	118GB for C drive and 931GB for D drive
Input	Keyboard and mouse
System type	64-bit operating system, x64-based processor

Table 3.3.2 Hardware Mobile Device for Mobile Application Development

Component	**Requirements**
Model	NTH-NX9 (HONOR 50)
Operating System	Android 13
Processor	Qualcomm Snapdragon 778G
RAM	8 GB + 2GB (HONOR RAM Turbo)
Storage	256 GB
System Type	Snapdragon ARM-based, 64-bit

ii. Software Requirements

Table 3.3.3 Software Component and Requirement for Mobile Application Development

Component	Requirements
Tools	**Visual Studio Code** The application requires a code editor that supports Flutter development. Visual Studio Code is selected due to its lightweight nature, wide plugin support, and compatibility with Flutter and Dart. It will be used throughout the project for writing source code, debugging, running emulators, and managing extensions like Flutter and Firebase integration.
	Git and Github Version control is used to manage the source code and track changes over time. Git will be used locally to manage commits and branches, while GitHub serves as the remote repository for backup, collaboration, and deployment version tracking.
	Android Studio Android SDK, build tools, and virtual devices for emulator-based testing when a physical device is not connected.
	Flutter DevTools The tools are needed to ensure efficient memory usage and stability. It will be used to identify memory leaks during development and monitor rendering performance.
	Firebase Console An interface used for managing the Firebase services. It also used to configure the authentication and monitor app usage analytics.
Languages, Libraries and Frameworks	**Dart Programming Language** Dart is selected as the core language for Flutter development because it can provide fast performance, ease of learning, and native compatibility with Flutter's architecture. Also, the mobile app must be developed using a language optimized for UI development and reactive programming.

	Flutter SDK Flutter is chosen due to its ability to compile into native code, support hot reload for faster development and provide a rich set of UI components suitable for child-friendly designs.
	Firebase Services Backend-as-a-Service platform providing authentication, such as email and password. It also includes a Firestore database for user profiles, reading history, and favorites. Also, it includes cloud storage for profile pictures, drawings, story images.
	Google Generative AI AI library for integrating Google's Gemini Vision API. Core technology for analyzing user drawing and generating story content from images. Provides advanced AI vision capabilities for image-to-text story generation.
	Flutter Test-to-Speech (TTS) TTS is a plugin package for Flutter. It provides system-level TTS capabilities for narrating stories, supporting speech rate control and voices. TTS also works with Audioplayers for audio playback management.
	Painter A drawing and painting library for Flutter. It provides custom drawing canvas functionality for the drawing module, enabling freehand drawing, shapes, and brush customization with undo and redo support.
	Image Handling Libraries The libraries include Image Picker which is used for selecting images from gallery, Gal is which used used for gallery access and saving, and Cached Network Image which is used for efficient image loading and caching. They used to manage user-generated content and story images.
	Local Storage and Utilities It is a shared reference for persistent user settings, preferences and the path provider for file system access.

3.4 Timeline

3.4.1 Overview

The timeline for this proposed project as shown in Figure 3.4.2.1 and Figure 3.4.2.2 are planned according to the methodology of the project as mentioned in Chapter 3.2. The timeline will be presented in the Gantt Chart to have an overview of the whole project including timeline for proposal writing, Final Year Project 1, and Final Year Project 2. First, the timeline started with reviewing the proposal writing title and discussed it with the supervisor. After confirming the topic and filling up the online undertaking form. The project started with defining problem statements and reviewed few of the existing applications to find out their strengths and weaknesses. Besides, the project scope and objectives identified. After the methodology used is defined for this project, the proposed project is summarized at the end of the report. The project timeline is also planned for Final Year Project 1 and Final Year Project 2 in order to ensure the project is complete on time.

In Final Year Project 1, the project followed an Agile sprint-based approach with 6 sprints over 7 weeks. The project setup in Flutter and Firebase was completed first, followed by the login and registration module within the first week. The drawing module with brush tools, color picker, and save functionality was developed in week 2, which was originally planned for Final Year Project 2 but was moved forward as it was essential for story generation. The story generation from the drawing module with genre selection and AI API integration was explored and developed in week 3. The image upload from gallery functionality was also implemented in week 3, allowing users to select images from their device gallery to generate stories. The story reading module with page navigation and progress tracking, along with the story editing module with text editor and page management, were developed in week 4. The text-to-speech integration module with voice selection, speed control, and word-by-word highlighting was developed in week 5, which was also originally planned for Final Year Project 2 but was identified as a core feature for the target age group. Story discovery features including search functionality, filter options, and favorites management, along with the profile and settings module with statistics tracking and configuration options, were also completed in week 5. Week 5 also included UI/ UX refinement, bug fixes, performance optimization, and comprehensive testing. Week 6 was dedicated to Final Year Project 1 report writing and submission. Week 7 was allocated for presentation preparation and the presentation to the supervisor and moderator.

In Final Year Project 2, the project followed an Agile sprint-based approach over 14 weeks. The first phase in Weeks 1 to 3 focused on the community sharing module, where story posting workflows, engagement features, and related community interactions were implemented. In Weeks 4 to 6, the development emphasis moved to safety and moderation, including moderation handling for community and library content to improve platform safety and content control. In Weeks 7 and 8, social feature extensions were developed, particularly friend-oriented communication flows and chat-related interaction support within the application. This was followed by Weeks 9 and 10, which focused on reader and literacy support enhancements to improve reading usability and child-friendly understanding features. Week 11 was allocated for performance and user-experience hardening, including optimization of loading behaviour, interaction smoothness, and reliability of integrated modules. Week 12 was dedicated to final integration and testing to ensure all components, including generation, reading, sharing, moderation, and social features, operated correctly as one complete system. Week 13 was allocated for Final Year Project 2 report writing and submission, while Week 14 was reserved for presentation preparation and final presentation to the supervisor and moderator. This timeline ensured that FYP2 progressed systematically from feature expansion to system stabilization, followed by documentation and assessment deliverables.

3.4.2 Gantt Chart

ID	Task Name	Start	End	Duration
1	Proposal Writing	2025-06-23	2025-09-28	98 days
2	Review Proposal Wr…	2025-06-23	2025-07-06	14 days
3	Confirm Topic Select…	2025-07-07	2025-07-20	14 days
4	Fill Up FYP Underta…	2025-07-21	2025-07-27	7 days
5	Define Problem Stat…	2025-07-28	2025-08-03	7 days
6	Literature Review	2025-08-04	2025-08-10	7 days
7	Define Project Scop…	2025-08-11	2025-08-17	7 days
8	Define Methodology…	2025-08-18	2025-08-24	7 days
9	Summarize Proposa…	2025-08-25	2025-08-31	7 days
10	Complete Project Ti…	2025-09-01	2025-09-07	7 days
11	Submission of Prop…	2025-09-08	2025-09-14	7 days
12	Proposal Writing Re…	2025-09-15	2025-09-21	7 days
13	Proposal Writing Pre…	2025-09-22	2025-09-28	7 days

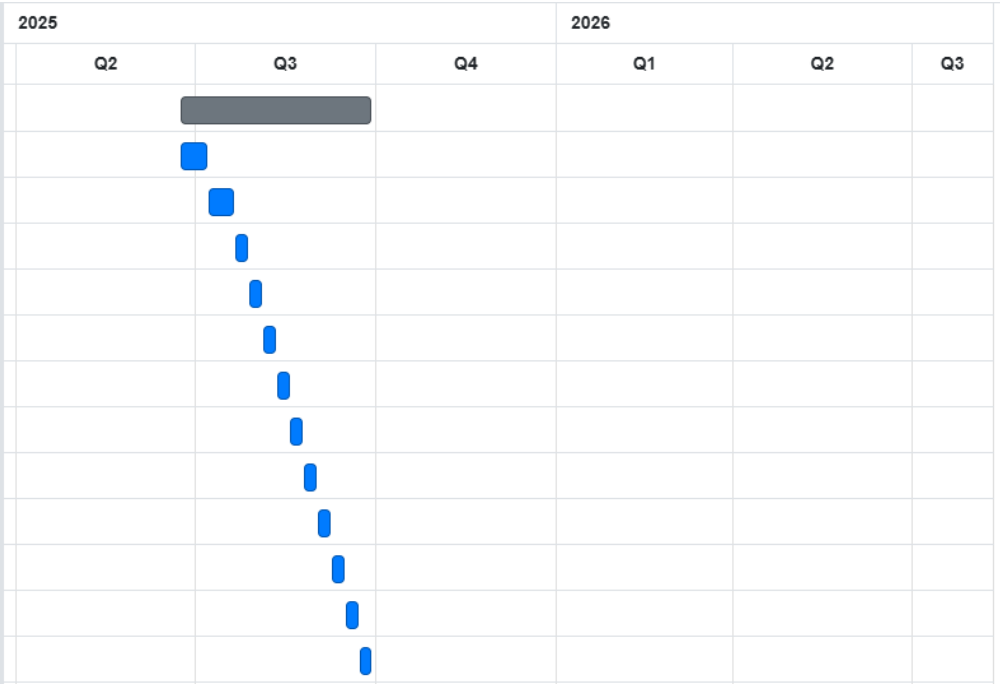


Figure 3.4.2.1 Gantt Chart of the Project Timeline

ID	Task Name	Start	End	Duration
14	FYP1	2025-11-03	2025-12-21	49 days
15	Project Setup and A…	2025-11-03	2025-11-09	7 days
16	Drawing Module	2025-11-10	2025-11-16	7 days
17	Story Generation an…	2025-11-17	2025-11-23	7 days
18	Story Reading and …	2025-11-24	2025-11-30	7 days
19	TTS, Discovery, Prof…	2025-12-01	2025-12-07	7 days
20	FYP1 Report Writing…	2025-12-08	2025-12-14	7 days
21	FYP1 Presentation	2025-12-15	2025-12-21	7 days
22	FYP2	2026-02-02	2026-05-10	98 days
23	Community Sharing …	2026-02-02	2026-02-22	21 days
24	Safety and Moderation	2026-02-23	2026-03-15	21 days
25	Social Feature Exte…	2026-03-16	2026-03-29	14 days
26	Reader and Literacy…	2026-03-30	2026-04-12	14 days
27	Performance and U…	2026-04-13	2026-04-19	7 days
28	Final Integration and…	2026-04-20	2026-04-26	7 days
29	FYP2 Report Writing…	2026-04-27	2026-05-03	7 days
30	FYP2 Presentation	2026-05-04	2026-05-10	7 days

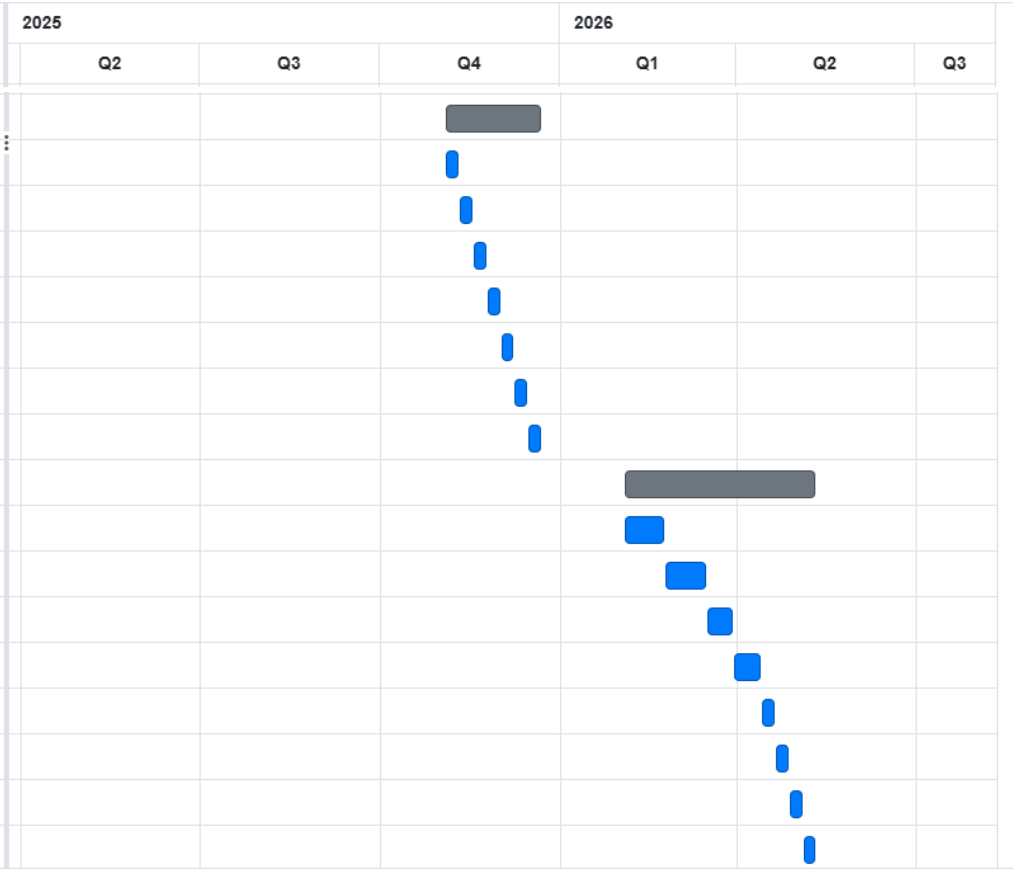


Figure 3.4.2.2 Gantt Chart of the Project Timeline (Continue)

3.5 Summary

All in all, this chapter explained the methodology used in this project, which is the Agile sprint-based methodology. This approach helped manage development in stages, allowed continuous testing and improvement, and supported changes to features during implementation. The hardware and software tools used to develop the front end and back end of the Story Craft application were also described in this chapter. In addition, the project timeline was planned and presented in a Gantt chart as shown in Figure 3.4.2.1 and Figure 3.4.2.2 to ensure proper time management and to complete all FYP2 tasks on schedule, including report submission and presentation.

CHAPTER 4 SYSTEM DESIGN

4.1 Overview

This chapter presents the system design of StoryCraft based on the implemented application. It explains how users interact with the system through authentication, drawing and AI story generation, reading and editing, profile management, and social features. Compared with FYP1, the FYP2 design extends the system with community sharing, moderation workflows, friend chat, and literacy support, so the chapter focuses on both functional flows and control flows that ensure content safety and reliable user experience. The use-case model is used to show actor interactions, while the use-case descriptions define preconditions, main events, and alternative outcomes for each major function.

4.2 Use-cases Diagram

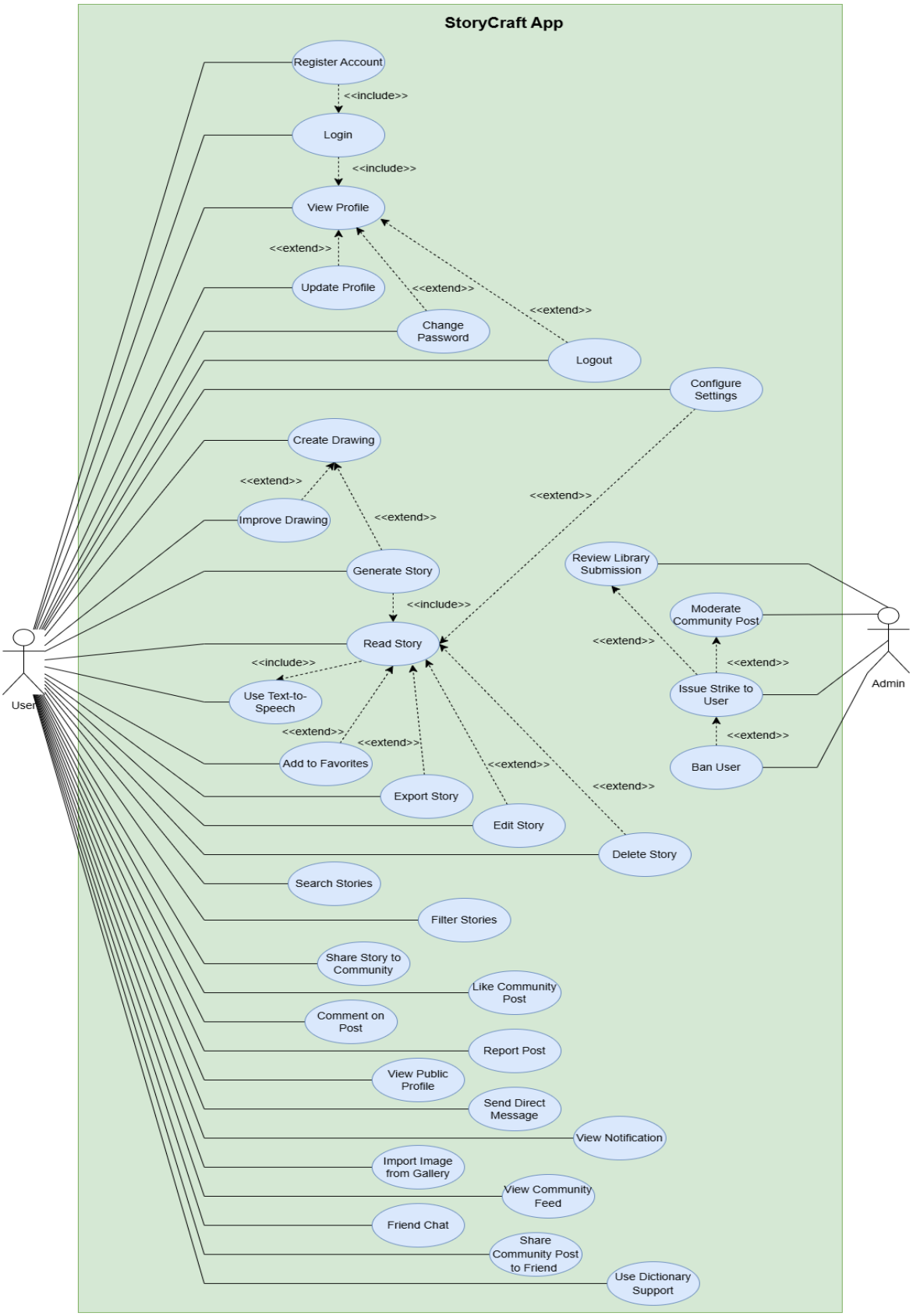


Figure 4.2.1 Use-case Diagram for Story Craft Mobile Application

4.3 Use-cases Description

Table 4.3.1 Register Account Use Case Diagram

Use Case ID	00001
Use Case Name	Register Account
Brief Description	The user creates a new account with a username, email and password.
Actor	User
Trigger	The user taps “Sign Up” on the login screen.
Precondition	User is not logged in. The app is on the login screen.
Normal flow of event	1. User taps the “Sign Up” button to switch to registration mode. 2. User enters a username, email, and password. 3. User enters the password again in the confirm password field. 4. User taps the “Sign Up” submit button. 5. System checks username availability. 6. System validates email format and password strength. 7. System creates a Firebase Authentication account. 8. System saves the username to Firestore. 9. System displays a success message in the UI. 10. System automatically switches back to login mode after 4 seconds.
Sub flows	2a. System shows username requirements when the username field is focused. 2b. System shows password requirements when the password field is focused. 5a. FirebaseUserDataService.isUsernameTaken checks if the username already exists in Firestore.

Alternative flows	5a. Username already taken: System shows "This username is already taken" error under the field. 6a. Invalid email format: Firebase returns invalid-email, system shows error in SnackBar. 6b. Passwords do not match: System shows "Passwords do not match" SnackBar. 7a. Email already in use: System shows "already a member" message in the UI. 7b. Network error: Firebase throws exception, system shows error SnackBar.

Table 4.3.2 Login Use Case Diagram

Use Case ID	00002
Use Case Name	Login
Brief Description	User logs into an existing account using email and password.
Actor	User
Trigger	User taps the "Login" button on the login screen.
Precondition	User is not logged in. User has an existing account.
Normal flow of event	1. User enters email and password on the login screen. 2. User taps the "Login" button. 3. System calls AuthService.signInWithEmailAndPassword with Firebase. 4. System checks if the account is permanently banned via CommunityService.isUserBanned. 5. System navigates to MainNavigationScreen.
Sub flows	4a. Load user's ban status from Firestore.
Alternative flows	3a. Wrong password: System shows "Incorrect email or password" below the password field. 3b. User not found: System shows "No account found with this email" error message.

	3c. Network error: Firebase throws exception, system shows error in the login field area. 4a. Account is banned: System signs out immediately and shows permanent ban message.

Table 4.3.3 Create Drawing Use Case Diagram

Use Case ID	00003
Use Case Name	Create Drawing
Brief Description	User creates a drawing on a digital canvas using drawing tools.
Actor	User
Trigger	User navigates to the draw tab in the main navigation.
Precondition	App is open. User may or may not be logged in.
Normal flow of event	1. User opens the drawing screen. 2. User selects a drawing tool: pen, marker, pencil, highlighter, or eraser. 3. User selects a colour from the colour palette. 4. User adjusts the brush size using the size slider. 5. User draws freehand strokes on the canvas. 6. User can place shapes: circle, square, triangle, star, or heart. 7. User can undo or redo strokes. 8. User taps “Save Drawing” to save the canvas. 9. System saves the drawing as a PNG image and JSON session file.
Sub flows	2a. Pen, marker, pencil, and highlighter tools apply different opacity and stroke styles. 6a. Shapes are placed and resized by drag gesture. 8a. Canvas is exported to an image file using Flutter’s RepaintBoundary.

Alternative flows	7a. Undo stack is empty: Undo button is disabled automatically. 8a. Save fails: System shows error message and allows retry.

Table 4.3.4 Improve Drawing Use Case Diagram

Use Case ID	00004
Use Case Name	Improve Drawing
Brief Description	User enhances their drawing using the AI-powered Drawing Enhancement Service.
Actor	User
Trigger	User taps the "Improve Drawing" button on the drawing screen.
Precondition	User has created or imported a drawing on the canvas.
Normal flow of event	1. User taps the "Improve Drawing" button. 2. System captures the current canvas as a high-resolution image. 3. System runs AI safety moderation on the image via LibrarySafetyModerationService using Gemini. 4. System displays an animated loading dialog. 5. System sends the image to DrawingEnhancementService. 6. DrawingEnhancementService calls the Gemini API to enhance the image. 7. System receives the enhanced image. 8. System saves both the original and enhanced versions. 9. System displays the enhanced drawing on the canvas.
Sub flows	2a. Canvas is captured via RepaintBoundary into a PNG byte buffer. 6a. Gemini API processes the image and returns an improved visual. 8a. Enhanced image is saved to local storage alongside the original.
Alternative flows	6a. API timeout: System shows error dialog and allows retry. 6b. API error or invalid response: System shows error message, original drawing is preserved.

	7a. Save fails: System shows error, original drawing remains unaffected.

Table 4.3.5 Generate Story from Drawing Use Case Diagram

Use Case ID	00005
Use Case Name	Generate Story from Drawing
Brief Description	User generates an AI story based on their drawing or imported image.
Actor	User
Trigger	User taps the "Generate Story" button on the drawing screen or story detail screen.
Precondition	User has a drawing (original or enhanced) or an imported image ready.
Normal flow of event	1. User taps "Generate Story". 2. System navigates to the Genre Selection screen. 3. User selects a genre: Fairy Tale, Adventure, Fantasy, Science Fiction, Mystery, Comedy, Nature, Friendship, or Any. 4. System captures the active drawing image. 5. System runs AI safety moderation on the image via LibrarySafetyModerationService using Gemini. 6. System displays an animated loading dialog. 7. System sends the image and selected genre to the story generation service. 8. System receives the generated story response. 9. System parses and formats the story into pages with title, description, and metadata. 10. System saves the story to local storage. 11. System navigates to story detail screen.
Sub flows	5a. Gemini analyses the image for safety before generation; if flagged, story is saved as pending. 7a. The API prompt includes genre instruction and image data.

	9a. Story title, description, categories, age range, and reading time are extracted from the AI response.
Alternative flows	5a. Image flagged by moderation: Story is saved with status "pending" and shown to user with a review notice. 7a. API timeout: System shows error dialog with retry option. 7b. API error: System shows error dialog and allows retry. 9a. Invalid story format: System retries generation automatically. 10a. Save fails: System shows error and offers retry.

Table 4.3.6 Read Story Use Case Diagram

Use Case ID	00006
Use Case Name	Read Story
Brief Description	User reads a story from the library or community using the story reader.
Actor	User
Trigger	User taps "Read Story" on a story detail screen.
Precondition	User is logged in. The story exists and has at least one page.
Normal flow of event	1. User taps the "Read Story" button on story detail screen. 2. System opens story detail screen with the story data. 3. System displays the first page with text and image. 4. User swipes left or right to navigate between pages. 5. System animates page transitions. 6. System updates reading progress after each page change. 7. User reaches the last page. 8. System saves final reading progress on exit.
Sub flows	3a. Page image is displayed above the text. 6a. ReadingProgressService stores the last page read per story ID. 6b. Progress percentage is calculated as currentPage or totalPages.
Alternative flows	2a. Story has no pages: System shows error and returns to previous screen. 3a. Image fails to load: System shows a placeholder icon.

	6a. Progress save fails: System retries in the background.

Table 4.3.7 Search Stories Use Case Diagram

Use Case ID	00007
Use Case Name	Search Stories
Brief Description	User searches for stories in the library by keyword.
Actor	User
Trigger	User taps the search icon on the library screen.
Precondition	User is on the library screen.
Normal flow of event	1. User taps the search icon in the library AppBar. 2. System reveals the search bar and focuses the text field. 3. User types a search keyword. 4. System filters stories in real-time as the user types. 5. System matches the keyword against story title, author, and category fields. 6. System displays only the matching stories 7. User taps a story card to open it. 8. User taps the X button or search icon again to close the search bar.
Sub flows	4a. Filtering is performed client-side on the already-loaded story list. 5a. Matching is case-insensitive using toLowerCase().
Alternative flows	5a. No stories match: System shows “No stories found” empty state with a search-off icon.

Table 4.3.8 Filter Stories Use Case Diagram

Use Case ID	00008
Use Case Name	Filter Stories
Brief Description	User filters the library story list by category type.
Actor	User
Trigger	User taps one of the filter buttons on the library screen.
Precondition	User is on the library screen.

Normal flow of event	1. User taps a filter button: All, Continue Reading, Favorites, Generated Stories, My Drawings, or Saved (Community). 2. System highlights the selected filter button in pink. 3. System applies the filter and reloads the story list. 4. System displays only stories matching the selected filter. 5. Filter state is maintained while user is on the library screen.
Sub flows	3a. "All": Shows all user-generated, discovered, favourited, and bookmarked community stories. 3b. "Continue Reading": Shows regular stories with reading progress that are not yet completed. 3c. "Favorites": Shows stories where ID is in the favourites list from ReadingProgressService. 3d. "Generated Stories": Shows stories whose ID starts with "ai-generated-". 3e. "My Drawing": Shows stories whose ID starts with 'drawing-'. 3f. "Saved (Community)": Shows stories with a non-null communityPostId.
Alternative flows	3a. No stories match the filter: System shows an appropriate empty state message per filter type.

Table 4.3.9 Add to Favorites Use Case Diagram

Use Case ID	00009
Use Case Name	Add to Favorites
Brief Description	User marks or unmarks a story as a favourite.
Actor	User
Trigger	User taps the heart icon on the story detail screen app bar.
Precondition	User is logged in and viewing a story detail screen.
Normal flow of event	1. User taps the heart icon on the story detail screen. 2. System checks the current favourite status via ReadingProgressService.isFavorite.

	3. If not favourited: System calls ReadingProgressService.addFavorite with the story ID. 4. If already favourited: System calls ReadingProgressService.removeFavorite. System updates the heart icon colour (red = favourited, outline = not favourited).
Sub flows	3a. Favourite IDs are stored via UserDataService to ensure account-specific persistence.
Alternative flows	3a. Save fails: System shows error and reverts the icon state.

Table 4.3.10 View Profile Use Case Diagram

Use Case ID	00010
Use Case Name	View Profile
Brief Description	User views their profile information, statistics, and account quick actions.
Actor	User
Trigger	User navigates to the profile tab in the main navigation.
Precondition	User is logged in.
Normal flow of event	1. User opens the profile screen. 2. System loads the user's profile data from Firestore. 3. System loads reading statistics: total stories read, drawings created, favourites count and reading time. 4. System displays the profile picture like avatar or custom image and username. 5. System displays a strike or bans warning banner if the user has community strikes. 6. System shows quick action buttons: Content Review, Community Posts, Settings, and others. 7. User can tap any quick action to navigate to the corresponding screen.
Sub flows	2a. Profile data includes username, email, profile picture, and moderation status.

	3a. Statistics are calculated from reading progress, story service, and favourites data. 5a. If strikeCount > 0, a yellow warning banner is shown. If banned, a red banner is shown.
Alternative flows	2a. Profile load fails: System shows error and a refresh button. 4a. Profile picture fails to load: System shows the default avatar.

Table 4.3.11 Update Profile Use Case Diagram

Use Case ID	00011
Use Case Name	Update Profile
Brief Description	User updates their username or profile picture.
Actor	User
Trigger	User taps the edit icon next to the username or profile picture on the profile screen.
Precondition	User is logged in and viewing their profile.
Normal flow of event	1. User taps the edit icon on the profile. 2. System displays an edit dialog for username or an avatar/gallery picker for the picture 3. User enters a new username or selects a new picture. 4. System validates the username format like minimum 3 characters, letters or numbers and . or _ only). 5. System checks username uniqueness via FirebaseUserDataService.isUsernameTaken. 6. System saves the new username via ProfileService.saveUsername or uploads the image to Firebase Storage. 7. System updates the Firestore user document. 8. System refreshes the profile display with the updated data.
Sub flowsu	2a. Username dialog shows the current username as the initial value. 2b. Picture picker shows avatar options and a “Choose from Gallery” option.

	6a. Profile picture is uploaded to Firebase Storage and the download URL is saved to Firestore.
Alternative flows	4a. Username too short: System shows validation error. 5a. Username taken: System shows "username already taken" error. 6a. Upload fails: System shows error and allows retry.

Table 4.3.12 Export Story Use Case Diagram

Use Case ID	00012
Use Case Name	Export Story
Brief Description	User exports the full story content as formatted text to the clipboard.
Actor	User
Trigger	User taps the "Export" button on story detail screen.
Precondition	User is logged in and viewing a story with at least one page.
Normal flow of event	1. User taps the "Export" button. 2. System builds a formatted text string with story title, description, author, categories, age range, reading time, and all page content with page numbers 3. System copies the formatted text to the device clipboard. 4. System opens a dialog showing the full exported text. 5. System displays a success message confirming the text was copied.
Sub flows	2a. Each page is prefixed with "PAGE N:" for clarity. 2b. Metadata section is formatted with emoji labels.
Alternative flows	1a. Story has no pages: System shows a SnackBar "This story has no pages to export".

Table 4.3.13 Change Password Use Case Diagram

Use Case ID	00013
Use Case Name	Change Password
Brief Description	User changes their account password from the profile screen.

Actor	User
Trigger	User taps "Change Password" in the profile quick actions.
Precondition	User is logged in with an email or password account.
Normal flow of event	1. User taps “Change Password”. 2. System shows a change password dialog. 3. User enters the current password, a new password, and confirms the new password. 4. System validates that the new password and confirm password match. 5. System validates new password strength like minimum 8 characters, uppercase, lowercase, and number. 6. System reauthenticates the user with the current password via Firebase Authentication. 7. System updates the password in Firebase Authentication. 8. System shows a success message.
Sub flows	6a. Firebase EmailAuthProvider.credential is used to reauthenticate the user before changing the password.
Alternative flows	4a. Passwords do not match: System shows “Passwords do not match” error. 5a. Weak password: System shows password requirement hint. 6a. Incorrect current password: Firebase throws wrong-password error, system displays error. 7a. Update fails: System shows error and allows retry.

Table 4.3.14 Edit Story Use Case Diagram

Use Case ID	00014
Use Case Name	Edit Story
Brief Description	User edits the title, description, and page content of a generated story.
Actor	User
Trigger	User taps “Edit Story” on story detail screen.

Precondition	User is logged in. The story's moderationStatus is not "pending". The user is the story owner.
Normal flow of event	1. User taps "Edit Story". 2. System opens story edit screen, loading the story's title, description, and all pages 3. System shows a one-time safety reminder dialog per session. 4. User modifies the title, description, or page text using text fields 5. User can add a new page, delete a page (minimum 1 page), or reorder pages using up or down arrow icons. 6. User taps the save icon or "Save Changes" button. 7. System validates that the title is not empty and at least one page has content. 8. System runs AI text moderation on the edited story via LibraryStoryTextModerationService 9. System shows an animated loading dialog during moderation. 10. System saves the updated story via GeneratedStoryService.updateStory. System shows a success dialog and navigates back to StoryDetailScreen.
Sub flows	5a. Deleting the last page shows a "story must have at least one page" error. 8a. Gemini analyses the text for safety; result sets moderationStatus, moderationRisk, and moderationTier.
Alternative flows	7a. Empty title: System shows error dialog "Please enter a story title". 7b. No page content: System shows error "Please add content to at least one page". 8a. Moderation fails: System defaults to pending status. 10a. Save fails: System shows error dialog and allows retry.

Table 4.3.15 Delete Story Use Case Diagram

Use Case ID	00015
Use Case Name	Delete Story
Brief Description	User permanently deletes a generated story from their library.
Actor	User
Trigger	User taps the “Delete” button on story detail screen.
Precondition	User is logged in. The user is the owner of the story. The story is not in “pending” or “rejected” moderation status.
Normal flow of event	1. User taps the “Delete” button. 2. System shows a confirmation dialog with the story title. 3. User confirms deletion. 4. System calls GeneratedStoryService.deleteStory to remove the story JSON from local storage. 5. System deletes the cover image file from local storage 6. System deletes drawing image files if they exist. 7. System removes the story from reading progress data via ReadingProgressService 8. System removes the story from the favourites list if it was favourited. 9. System shows a success message and navigates back to the library screen
Sub flows	5a. Cover image path is checked; if it exists as a local file, File.delete() is called. 6a. Drawing session JSON file is also deleted if present.
Alternative flows	3a. User cancels the dialog: No action is taken, dialog closes. 4a. Delete fails: System shows error and allows retry. 5a. Image deletion fails: System logs a warning and continues (non-blocking).

Table 4.3.16 Configure Settings Use Case Diagram

Use Case ID	00016
Use Case Name	Configure Settings
Brief Description	User configures reading preferences while reading a story, including voice, speed, night mode, and text size.
Actor	User
Trigger	User taps the voice, speed, or night mode button in the StoryReaderScreen toolbar.
Precondition	User is reading a story in story reader screen.
Normal flow of event	1. User taps a setting button in the reader toolbar. 2. System displays the corresponding settings dialog or toggles the mode immediately. 3. For voice: System shows a voice selection dialog with available TTS voices. 4. For speed: System shows a speed selection dialog with options 0.5x, 0.75x, 1.0x, 1.25x. 5. For night mode: System toggles dark or light theme immediately. 6. User adjusts text size via pinch-to-zoom gesture on the page. 7. System saves the selected preferences via SharedPreferences. 8. System applies the settings immediately to the current reading session.
Sub flows	3a. Voice options are loaded from the device's available TTS engine voices. 7a. Settings are persisted across sessions using SharedPreferences keys.
Alternative flows	3a. No voices available: System shows a "TTS not available" message. 7a. Save fails: System shows a warning; settings apply for the session but may not persist.

Table 4.3.17 Use Text-to-Speech Use Case Diagram

Use Case ID	00017
Use Case Name	Use Text-to-Speech (TTS)
Brief Description	User listens to an AI-narrated reading of the story with word highlighting and auto-scroll.
Actor	User
Trigger	User taps the TTS play button while reading a story in story reader screen.
Precondition	User is reading a story. TTS engine is available on the device.
Normal flow of event	1. User taps the TTS play button 2. System loads TTS preferences: voice provider, voice selection, and speech rate from SharedPreferences 3. System initialises the TTS engine. 4. System starts narrating the text of the current page from the beginning. 5. System highlights words in real-time as they are spoken 6. System auto-scrolls the page to keep the currently spoken word visible. 7. User can tap the pause button to pause and resume narration. 8. User can tap the stop button to stop narration. 9. System saves the playback position when the user exits.
Sub flows	2a. If Cloud TTS is enabled, the system uses the selected cloud voice. Otherwise, it falls back to the system TTS. 5a. Word boundary callbacks from the TTS engine are used to sync the highlight index.
Alternative flows	3a. TTS initialisation fails: System shows error and suggests checking device TTS settings. 4a. Text conversion error: System shows error and allows retry. 5a. Highlighting goes out of sync: System re-syncs on the next sentence. 7a. Audio playback error mid-narration: System shows error and allows the user to retry.

Table 4.3.18 Logout Use Case Diagram

Use Case ID	00018
Use Case Name	Logout
Brief Description	User logs out of their account while preserving local story data.
Actor	User
Trigger	User taps the “Logout” button on the Profile screen.
Precondition	User is logged in.
Normal flow of event	1. User taps the “Logout” button on the Profile screen 2. System displays a confirmation dialog informing the user that local data will be preserved. 3. User confirms the logout. 4. System calls AuthService.signOut to sign out from Firebase Authentication 5. System clears session-specific flags. 6. System navigates to the login screen. 7. Local story data, drawings, and reading progress are preserved on the device.
Sub flows	4a. Firebase Auth.signOut clears the current user session token. 5a. Session flags stored in SharedPreferences are cleared.
Alternative flows	3a. User cancels: Dialog closes, user remains logged in. 4a. Sign-out fails: System shows an error message and allows retry.

Table 4.3.19 Import Image from Gallery Use Case Diagram

Use Case ID	00019
Use Case Name	Import Image from Gallery
Brief Description	User imports an image from the device photo gallery to generate a story from it.
Actor	User
Trigger	User taps the “Import from Gallery” icon button on the drawing screen.

Precondition	User is on the drawing screen.
Normal flow of event	1. User taps the “Import from Gallery” button. 2. System opens the device image picker. 3. User selects an image from the gallery. 4. System compresses and resizes the image. 5. System saves the image to local storage as a new drawing entry via GeneratedStoryService. 6. System navigates to genre selection screen with the imported image. 7. User selects a story genre. 8. System runs AI safety moderation on the image via LibrarySafetyModerationService. 9. System generates a story from the image and selected genre. 10. System saves the generated story and navigates to story detail screen.
Sub flows	4a. Image is decoded and re-encoded as JPEG using Flutter’s image library. 8a. Gemini AI analyses the image; if flagged, story is saved as “pending” for admin review.
Alternative flows	3a. User cancels the picker: System returns to drawing screen with no action. 8a. Image flagged by moderation: Story is saved as “pending” and user is shown a review notice. 9a. API error: System shows error dialog with retry option.

Table 4.3.20 Share Story to Community Use Case Diagram

Use Case ID	00020
Use Case Name	Share Story to Community
Brief Description	User publishes an approved story to the public community feed.
Actor	User
Trigger	User taps “Share to Community” on story detail screen.

Precondition	User is logged in. The story's moderationStatus is "approved" in the library. User is not currently banned from the community.
Normal flow of event	1. User taps the "Share to Community" button. 2. System verifies that the user is not banned via CommunityService.canUserPost. 3. System checks the story's existing moderation metadata. 4. Inherited Trust Logic: Since the story is already library-approved, the system bypasses the community AI scan. 5. System creates a community post using the existing library safety scores. 6. Post is published live immediately to the community feed. 7. System shows a success notice via CommunityNotice. 8. System sends a notification to the user confirming the post is live.
Sub flows	-
Alternative flows	2a. User is banned: System shows a ban message and prevents sharing. 3a. Story not approved: System informs the user that the story must pass library moderation first. 4a. Inherited Trust Fail: If library metadata is missing, the system defaults to a "Pending" state for admin review.

Table 4.3.21 Like Community Post Use Case Diagram

Use Case ID	00021
Use Case Name	Like Community Post
Brief Description	User likes or un-likes a post on the community feed.
Actor	User
Trigger	User taps "Share to Community" on story detail screen.
Precondition	User is logged in and viewing the community screen or my community feed screen.
Normal flow of event	1. User taps the like icon on a post.

	2. System checks if the user has already liked the post from the local cache. 3. If not liked: System calls CommunityService to add the user's ID to the post's likes array in Firestore and increments the like count. 4. If already liked: System calls CommunityService to remove the user's ID and decrements the like count. 5. System updates the like icon state immediately for optimistic UI update. 6. System sends a "like" type notification to the post author via Firestore.
Sub flows	2a. Like status is checked from the CommunityPost.likedBy list. 6a. Notification is written to the author's notifications sub-collection in Firestore.
Alternative flows	2a. User is not logged in: System shows a login prompt. 3a. Firestore update fails: System shows an error and reverts the like icon state.

Table 4.3.22 Comment on Post Use Case Diagram

Use Case ID	00022
Use Case Name	Comment on Post
Brief Description	User submits a comment on a community story post, which is checked by AI moderation before publishing.
Actor	User
Trigger	User taps the comment icon on a post, opening the comments bottom sheet.
Precondition	User is logged in and viewing the community feed or a post detail.
Normal flow of event	1. User taps the comment icon on a post. 2. System opens the comments bottom sheet and loads existing comments in real-time from Firestore. 3. User types a comment in the text field. 4. User taps the send button.

	5. System first checks the comment against CommunityKeywordGate for banned keywords. 6. If the keyword gate passes, System sends the comment to CommunityAIModerationService for contextual analysis. 7. If AI approves: System saves the comment to Firestore and displays it. 8. System sends a "comment" type notification to the post author. 9. System updates the comment count on the post.
Sub flows	5a. CommunityKeywordGate performs a fast local keyword scan before calling the AI. 6a. Gemini analyses context, obfuscated profanity, and harmful intent. 8a. If the comment mentions another user with @username, a "tagged" notification is also sent.
Alternative flows	3a. Empty comment: The send button remains disabled. 5a. Banned keyword detected: Comment is blocked immediately with a safety warning. 6a. AI flags the comment: Comment is blocked with a safety warning to the user. 7a. Firestore save fails: System shows an error and allows retry.

Table 4.3.23 Report Post Use Case Diagram

Use Case ID	00023
Use Case Name	Report Post
Brief Description	User reports a community post for inappropriate or unsafe content.
Actor	User
Trigger	User taps the report icon on a community post card.
Precondition	User is logged in and viewing the community feed.
Normal flow of event	1. User taps the report icon on a post. 2. System checks if the user has already reported this post.

	3. System displays a report reason dialog with selectable options. 4. User selects a reason and confirms. 5. System creates a report document in Firestore linked to the post via CommunityService. 6. System flags the post as reported, adding it to the community moderation screen queue. 7. System sends a "reported" type notification to the post author. 8. System shows a confirmation message to the reporting user.
Sub flows	5a. The report document stores the reporter's user ID, reason, and timestamp. 6a. The community post document is updated with a reportedAt timestamp and reportReason field.
Alternative flows	2a. User has already reported this post: System shows an "already reported" notice. 4a. User cancels the dialog: No action is taken. 5a. Firestore write fails: System shows an error message.

Table 4.3.24 View Public Profile Use Case Diagram

Use Case ID	00024
Use Case Name	View Public Profile
Brief Description	User views another community member's public profile and their shared story posts.
Actor	User
Trigger	User taps on an author's name or avatar on a community post card.
Precondition	User is logged in and viewing the community feed.
Normal flow of event	1. User taps on the author's name or avatar. 2. System navigates to the public profile screen.

	3. System calls CommunityService.getPublicProfile to load the author's username and profile picture from Firestore. 4. System loads the author's community posts via my community feed screen with the author's user ID. 5. System displays the username, profile picture, and the author's approved posts. 6. User can tap on a post to read the story in story detail screen. 7. User can tap "Message" to open a direct message conversation with the author.
Sub flows	3a. Only public data like username and profile picture is fetched — private data is not exposed. 4a. The feed shows only the author's approved community posts.
Alternative flows	3a. Profile data not found: System shows "User not found" message. 3b. Network error: System shows error with a retry option.

Table 4.3.25 Send Direct Message Use Case Diagram

Use Case ID	00025
Use Case Name	Send Direct Message
Brief Description	User sends a private direct message to another community member, optionally sharing a community post.
Actor	User
Trigger	User taps the Messages FAB on the Community screen or the "Message" button on a public profile.
Precondition	User is logged in.
Normal flow of event	1. User opens the Messages Inbox. 2. System loads all existing conversations in real-time via FriendChatService.conversationsStream, ordered by latest activity.

	3. System shows each conversation with the peer's username, profile picture, last message preview, and unread count badge. 4. User taps an existing conversation or starts a new one from a public profile or post share. 5. System opens friend chat screen for the selected peer. 6. User types a message or shares a community post link. 7. User taps the send button. 8. System saves the message to Firestore. 9. System sends a "shared_post" notification to the recipient. 10. System updates the unread message badge in real-time.
Sub flows	2a. Conversations are stored in Firestore as shared documents between two user IDs. 8a. Each message document stores senderId, text, optional postId, and timestamp.
Alternative flows	6a. Empty message with no shared post: Send button is disabled. 8a. Firestore write fails: System shows an error and allows retry.

Table 4.3.26 View Notifications Use Case Diagram

Use Case ID	00026
Use Case Name	View Notifications
Brief Description	User views all in-app activity notifications including likes, comments, moderation updates, and direct messages.
Actor	User
Trigger	User taps the notification bell icon on the home or community screen.
Precondition	User is logged in.
Normal flow of event	1. User taps the notification bell. 2. System navigates to community notifications screen. 3. System loads notifications in real-time via CommunityService.userNotificationsStream, ordered newest first.

	4. Unread notifications are shown with a purple background; read ones are grey. 5. User taps a notification to navigate to the relevant content. 6. System marks the tapped notification as read. 7. User can tap "Mark all read" to dismiss all unread indicators at once.
Sub flows	5a. "comment" or "tagged": Navigates to my community feed with the comment sheet open. 5b. "like" or "approved" or "report_dismissed": Navigates to my community feed screen with the post highlighted. 5c. "library_approved" or "library_rejected": Navigates to story detail screen for that story. 5d. "shared_post": Navigates to friend chat screen with the sender. 5e. "submitted" or "rejected": Navigates to story detail screen for the submitted post.
Alternative flows	3a. Network error: System shows "Could not load notifications" message. 3b. No notifications yet: System shows empty state with bell icon and "No notifications yet" label.

Table 4.3.27 Review Library Submissions Use Case Diagram

Use Case ID	00027
Use Case Name	Review Library Submissions
Brief Description	Admin reviews user drawings and AI-generated stories pending approval in the library moderation queue.
Actor	Admin
Trigger	Admin taps "Moderate Library" from the admin panel on the Profile screen.
Precondition	User is logged in as an Admin.
Normal flow of event	1. Admin opens library moderation screen.

	2. System loads all queue items into three tabs: Pending, Approved, and Rejected. 3. System displays each item with cover thumbnail, owner email, strike count, AI risk score percentage, moderation tier, Gemini screening notes, and summary reason. 4. Admin reviews the content in the Pending tab. 5. Admin taps "Approve": System calls LibraryModerationQueueService.approveItem, sets moderationStatus to "approved" in Firestore, and sends a "library_approved" notification to the owner. 6. Admin taps "Reject": System shows a rejection reason dialog with Gemini screening notes pre-filled. Admin enters a reason and confirms. System calls LibraryModerationQueueService.rejectItem, sets moderationStatus to "rejected", increments the author's strike count, and sends a "library_rejected" notification. 7. System refreshes the queue after each action.
Sub flows	3a. AI risk score is shown as a percentage like "Risk 35% → pending admin review (20–49%)"). 6a. Rejecting increments strikeCount in the user's Firestore moderation document via CommunityService.
Alternative flows	2a. Load fails: System shows an error with a retry button. 5a. Approve action fails: System shows error notice and the item remains in pending. 6a. Admin cancels the reject dialog: No action is taken, item remains in pending.

Table 4.3.28 Moderate Community Posts Use Case Diagram

Use Case ID	00028
Use Case Name	Moderate Community Posts
Brief Description	Admin reviews and takes action on community posts that have been reported by users.

Actor	Admin
Trigger	Admin taps “Moderate Community” from the admin panel on the profile screen.
Precondition	User is logged in as an Admin.
Normal flow of event	1. Admin opens community moderation screen. 2. System loads all pending reported posts from Firestore. 3. System displays each reported post with cover image, story title, author name, author email, current strike count, report reason, and AI or keyword screening analysis. 4. Admin taps “Preview” to open the story in story reader screen and read the full content. 5. Admin taps “Hide Post”: System calls CommunityService.hideReportedPost, removes the post from the public community feed, and shows a success notice. 6. Admin taps “Keep Visible”: System calls CommunityService.dismissReportedPost, clears the report flag, and the post remains visible in the community. 7. System refreshes the reported post list after each action.
Sub flows	2a. Reports are fetched from a “reports” sub-collection, filtered to status = “pending”. 3a. AI screening reasoned and ML flags are both displayed for each report.
Alternative flows	2a. Load fails: System shows error with a retry option. 5a. Hide action fails: System shows “Failed to hide post” error notice. 6a. Dismiss action fails: System shows “Failed to dismiss report” error notice.

Table 4.3.29 Issue Strike to User Use Case Diagram

Use Case ID	00029
Use Case Name	Issue Strike to User
Brief Description	A community safety strike is issued to a user when an admin rejects their library submission or hides their reported community post. Strikes are tracked and lead to an automatic ban after 3 strikes.
Actor	Admin
Trigger	Admin confirms a “Reject” action on a library item or a “Hide Post” action on a reported community post.
Precondition	Admin is logged in. A reject or hide action has been confirmed on the target content.
Normal flow of event	1. Admin confirms a rejection or hides action in the moderation screen. 2. System calls CommunityService to increment the author’s strikeCount in their Firestore moderation document by 1. 3. System reads the updated strike count and compares it against the maximum (strikesBeforeBan = 3). 4. System writes a strike warning notification to the user’s Firestore notifications sub-collection. 5. The updated strike count and ban status are shown in the moderation screen’s author info line. 6. If the strike count reaches 3: System sets bannedUntil to a future date in Firestore, permanently restricting the user from posting. 7. On the user’s next profile view, the strike count and ban warning banner are displayed on the profile screen.
Sub flows	2a. CommunityService.getUserModeration loads the current strike document for the author. 6a. The ban is enforced at the start of every community post and share action via CommunityService.canUserPost. 7a. Profile screen reads the strike count and ban date from Firestore and shows a coloured warning banner.

Alternative flows	2a. Firestore update fails: System shows an error notice, and the strike is not recorded. 6a. User is already banned: System logs a warning but does not apply a duplicate ban.

Table 4.3.30 View Community Feed Use Case Diagram

Use Case ID	00030
Use Case Name	View Community Feed
Brief Description	User browses stories shared by other members in the public community.
Actor	User
Trigger	User taps the “Community” tab in the bottom navigation.
Precondition	User is logged in..
Normal flow of event	1. User taps the “Community” tab. 2. System loads approved community posts from Firestore in real-time. 3. System displays posts as cards with cover image, title, author, and likes or comments count. 4. User scrolls through the feed. 5. User can pull down to refresh the feed manually. 6. User can tap the “Sort” icon to change the order.
Sub flows	2a. Calls CommunityService.getApprovedPostsStream(). 6a. Sorts posts based on createdAt or likesCount.
Alternative flows	2a. No posts available: System shows “No posts found” message. 2b. Network error: System shows error SnackBar and allows manual refresh.

Table 4.3.31 Friend Chat Use Case Diagram

Use Case ID	00031
Use Case Name	Friend Chat
Brief Description	User engages in a real-time text conversation with a friend.
Actor	User
Trigger	User selects a conversation from the Messages Inbox.
Precondition	User is logged in and has an active conversation.
Normal flow of event	1. User enters the friend chat screen. 2. System loads the message history from Firestore. 3. System listens for new messages in real-time. 4. User types and sends a message. 5. System displays the new message immediately for both users. 6. System updates the "last message" preview in the inbox. 7. System marks incoming messages as "read" when the chat is open.
Sub flows	3a. Uses a Firestore Stream to listen to the messages sub-collection. 7a. Updates unreadCount for the specific user in the conversation document.
Alternative flows	2a. Chat history is empty: System shows "Start a conversation" hint. 4a. Message failed to send: System shows error icon next to message, allows retry.

Table 4.3.32 Share Community Post to Friend Use Case Diagram

Use Case ID	00032
Use Case Name	Share Community Post to Friend
Brief Description	User shares an existing community post with a friend via Direct Message.
Actor	User
Trigger	User taps the “Share” paper plane icon on a community post.
Precondition	User is logged in. The post is approved and visible.
Normal flow of event	1. User taps the Share icon on a post. 2. System opens the user friend screen. 3. User selects a friend from the list. 4. System creates a new message containing the postId and a preview of the story. 5. System navigates the user into the chat with that friend. 6. Recipient receives a notification with the shared post preview.
Sub flows	4a. Calls FriendChatService.sendMessage with a non-null postId. 6a. Sends a “shared_post” type notification to the friend.
Alternative flows	3a. User has no friends: System suggests sharing stories to start conversations. 4a. Post no longer exists: System shows “Post unavailable” warning.

Table 4.3.33 Use Dictionary Support Use Case Diagram

Use Case ID	00033
Use Case Name	Use Dictionary Support
Brief Description	User looks up the definition of a difficult word while reading a story.
Actor	User
Trigger	User long-presses or double-taps a word in the story reader.
Precondition	User is reading a story in story reader screen.
Normal flow of event	1. User selects a specific word in the story text. 2. System calls DictionaryService to fetch the definition. 3. System displays the word definition, phonetics, and example in a bottom sheet or dialog. 4. User dismisses the definition to continue reading.
Sub flows	2a. Calls an external Dictionary API or uses a local cache for common words. 3a. Formats the API response into a child-friendly layout.
Alternative flows	2a. Word not found: System shows “Definition not found” message. 2b. No internet: System shows “Offline: message if the dictionary requires a connection. 2a. Post no longer exists: System shows “Post unavailable” warning.

Table 4.3.34 Ban User Use Case Diagram

Use Case ID	00034
Use Case Name	Ban User
Brief Description	System permanently restricts a user community interaction after reaching the 3-strike count.
Actor	Admin
Trigger	A user receives their 3rd strike during content moderation.
Precondition	User has 2 existing strikes in their moderation document,
Normal flow of event	1. Admin rejects content or hides a post, triggering a strike record. 2. System increments strikeCount to 3. 3. Automatic Enforcement: System sets the banned field to true in the user's moderation record. 4. Community Cleanup: System automatically hides all previous posts by this author from the public feed to ensure platform safety. 5. System records the lastWarningMessage explaining the permanent block. 6. System displays a "Permanently Blocked" notice to the user upon their next login attempt.
Sub flows	-
Alternative flows	1a. Early Ban: An administrator can manually flag an account for a permanent ban even if they have fewer than 3 strikes for extreme violations. 4a. Cleanup Fail: If the bulk-hide of posts fails, the system logs an error but maintains the user's banned status.

4.4 Project Development

4.4.1 Frontend Development

The StoryCraft application frontend was developed using Flutter, a cross-platform mobile development framework developed by Google. Flutter enables the development of native applications for Android platforms from a single codebase, written in the Dart programming language. This choice provides significant advantages including code reusability, consistent user experience across platforms, and rapid development through hot reload capabilities.

The application utilizes a comprehensive set of Flutter packages and dependencies managed through pubspec.yaml as shown in Figure 4.4.1.1. Key dependencies include firebase services for backend functionality, such as firebase_core, firebase_auth, firebase_remote_config, cloud_firestore, and firestore_storage; AI service integrations, such as google_generative_ai and http; text-to-speech capabilities, such as flutter_tts and audioplayers; image handling, such as cached_network_image, image_picker, image, and gal; local storage, such as shared_preferences and path_provider; community and sharing features, such as share_plus and app_links; reader utilities, such as scrollable_positioned_list; and utility packages for logging and package information, such as logger and package_info_plus.

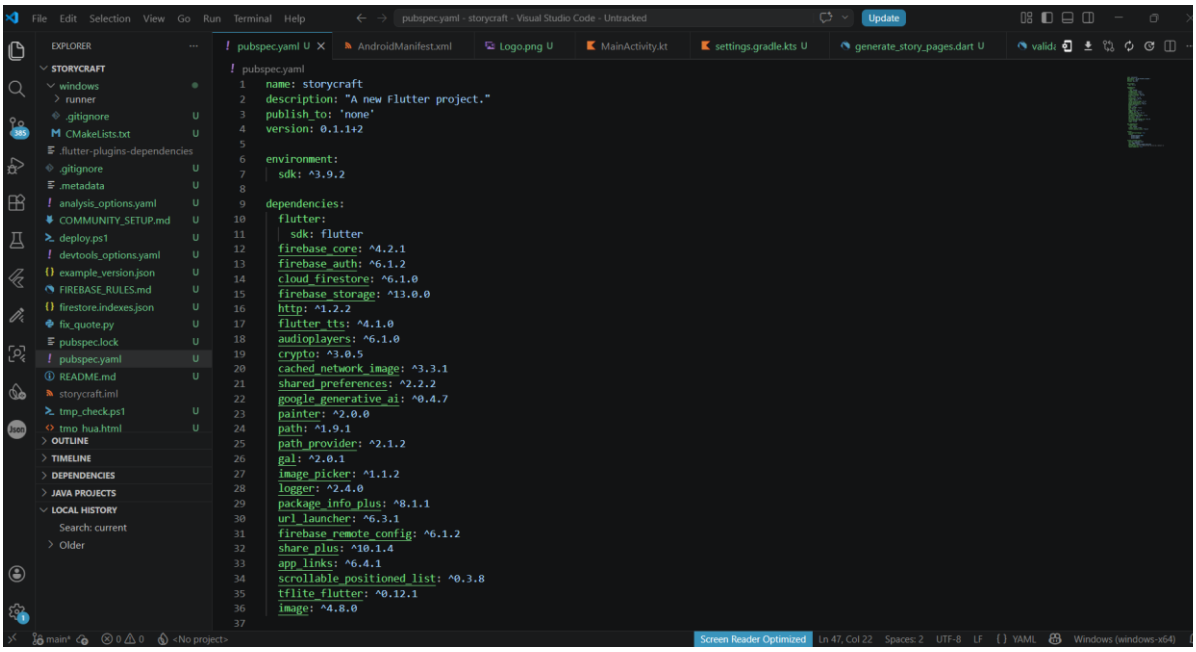


Figure 4.4.1.1 Pubspec.yaml

The frontend architecture follows the Model-View-Service (MVS) pattern, which separates concerns into distinct layers to ensure clean code organization and maintainability. The model

layer encompasses data structures that represent application entities such as Story, StoryPage, CommunityPost, Comment, and Notification, providing a consistent data representation throughout the application. The views layer, implemented through screens in Flutter, handles all user interface components that display data and manage user interactions, ensuring a clear separation between presentation logic and business logic. The services layer encapsulates business logic and data access operations, handling critical functionality including user authentication AI-powered content moderation, real-time community interactions and automated data migration. This architectural pattern ensures maintainability, testability, and scalability of the codebase, allowing for easy modification of individual components without affecting the entire system, and facilitates the handling of dynamic community data via asynchronous streams.

The application consists of 20 main screens implemented as Flutter widgets: splash screen, login screen, main navigation screen, home screen, library screen, drawing screen, genre selection screen, story detail screen, story reader screen, story edit screen, profile screen, community screen, community notification screen, friends inbox screen, friend chat screen, public profile scree, library moderation screen, community moderation screen, my community feed screen, and user friend screen. Each screen is designed with intuitive navigation and age-appropriate design elements.

The application employs a hybrid state management approach combining Flutter's setState for local component state, firebase authentication reactive streams for global authentication state, and SharedPreferences for persistent local storage of user preferences and settings. In FYP2, the state management approach has been extended to incorporate Firestore reals time streams using StreamBuilder widgets, which enable the community feed, direct messaging inbox, and notification badge to update instantly without requiring manual page refreshes. The service layer manages data flow and state synchronization between components while maintaining separation of concerns.

The application utilizes Flutter's material design 3 component library for visual consistency. Navigation is handled through BottomNavigationBar, AppBar, TabBar with TabBarView for administrative moderation queues. Display components include GridView and ListView for story collections with lazy loading, CircleAvatar for user profiles, and Badge for real-time notification alerts. A custom StoryCard widget for reusable story previews, and PageView for page-by-page reading. User input is managed through TextFormField, ElevatedButton, Slider,

and a custom ColorPicker. Feedback mechanisms have been expanded to include ModalBottomSheet for community comments and dictionary definitions, Snackbar for system messages, Dialog for confirmations, and StreamBuilder for real-time synchronization of likes and messages.

The drawing screen implements a custom drawing canvas using Flutter's CustomPaint widget with a CustomPainter class for rendering operations. Touch input is captured through GestureDetector widgets that translate user interactions into drawing commands. The implementation includes brush tools, geometric shapes, an eraser, and an undo or redo system using stack-based data structures. Completed drawings are captured as high-resolution images optimized for processing. The drawing screen also supports image import from the device gallery. Users may select an existing photo from their device as a story prompt, which is then compressed and optimized before passed to the story generation pipeline, broadening the creative options available to younger users.

The TTS functionality integrates multiple providers, such as System TTS for offline functionality, Google Cloud TTS for premium voice quality, and Amazon Polly for specialized children's voices. Word-by-word highlighting synchronizes with audio playback through position callbacks and word boundary detection, helping children follow along with narration. Playback controls include play, pause, stop, and speed adjustment from 0.5x to 1.25x. The reading experience has been enhanced with the addition of dictionary support. Users may interact with any word in the story reader to instantly retrieve its definition and phonetics from the DictionaryService. The definition is displayed in a child-friendly bottom sheet dialog, enriching the literacy and vocabulary development aspects of the application.

The application implements responsive design using constraint-based widgets, such as Expanded, Flexible, and SizedBox and Flutter's MediaQuery API for device-specific adaptations. The design supports both portrait and landscape orientations, with touch targets meeting accessibility guidelines for the target age group.

4.4.2 Backend Development

The StoryCraft application employs a Backend-as-a-Service (BaaS) architecture using Firebase, Google's comprehensive cloud platform. This architecture has been expanded to support complex community infrastructure, providing scalable infrastructure, real-time social capabilities, and integrated security features.

The backend architecture leverages multiple Firebase services for cloud infrastructure. Firebase authentication provides secure email and password authentication with built-in password hashing and token management. Cloud Firestore serves as the primary database using a NoSQL document-based architecture with real-time synchronization capabilities, managing new collections for community posts, comments, likes, notifications, and administrative moderation reports. Security is enforced through granular Firestore security rules that implement user-specific data access control and restrict administrative functions to verify admin accounts. Firebase storage provides scalable cloud-based file storage for profile pictures and user-generated drawing images with secure access control. Firebase remote config enables dynamic application configuration for version management and update notifications.

Data security is enforced through Firebase Security Rules, which define access permissions at the document level. For example, rules are configured to allow only the original author to update or delete their comments, while the moderation collection is restricted to "read-only" for users and "write" for administrative accounts.

The application utilizes a document-oriented NoSQL structure in Cloud Firestore. To ensure a developer could rebuild the system, the primary collections and their structures are defined as below Table 4.4.2.1.

Table 4.4.2.1 Primary Collection and Structure

Collection Path	Structure
admin	A simple collection used for security. If a user's ID exists in this collection, the app treats them as an Administrator.
community_posts	All the stories that users have shared. Can see the fields like status approved or pending.
friend_conservation	A collection stores the chat threads between two friends

library_moderation_queue	The queue for the Admin to approve or reject drawings.
public_profiles	A collection used to quickly show usernames and avatars in the community feed.
reported_posts	A list of posts that users have flagged for inappropriate content.
users	A list of all user IDs. Inside each one, can see username and email.

The application implements hybrid storage architecture combining cloud-based and local storage solutions. Cloud storage through firestore is utilized for user profiles, authentication data, reading progress tracking, favorites lists, and all community-related interactions that require synchronization across devices. Local storage through SharedPreferences provides persistent storage for app preferences and TTS settings. A key backend feature is the migration service, which coordinates the transfer of locally stored anonymous drawings and stories into the user's cloud-linked account upon registration, ensuring complete data persistence. File system storage handles larger data files including user-generated stories stored as JSON files and drawing images stored for optimal performance.

The application integrates with AI service providers using a multi-provider architecture for both content generation and real-time safety moderation. Google Gemini 1.5 Flash serves as the primary provider, leveraging advanced vision and language capabilities to analyze drawings, generate genre-specific stories, and evaluate community content for safety violations. The service architecture implements a multi-tier safety gate where Gemini analyzes image and text for risk; content with low risk is auto approved immediately, while medium-risk and high-risk items are queued for human review. OpenAI API integration provides a fallback story generation provider, with comprehensive error handling and retry mechanisms to ensure continuous availability. DALL-E 3 API integration provides image enhancement functionality that generates enhanced versions of user drawings. The service architecture implements primary and fallback providers with comprehensive error handling, retry mechanisms, exponential backoff strategies, and rate limiting management to ensure continuous availability.

The backend logic is systematically encapsulated in dedicated service classes that abstract Firebase and API interactions, providing a clean separation between business logic and external

service dependencies. The application consists of 21 service classes, expanded from 13 in FYP1 to handle the new moderation and social modules.

Authentication service shown in Figure 4.4.2.2 manages all authentication-related operations including user registration with email or password validation and profile initialization, login processes that verify credentials and establish user sessions, logout functionality that securely terminates sessions, and provides authentication state streams that allow UI components to reactively update when authentication status changes. The service has been enhanced to include mandatory community ban checks during the login process, verifying a user's moderation status via Firestore to prevent blocked accounts from accessing the community features.

User data service shown in Figure 4.4.2.3 implements critical data isolation mechanisms that ensure each user's data remains separate and secure. It provides account-specific data isolation through user ID-based data segregation, ensuring that data access is restricted to the appropriate user account. The service has been integrated with FireBaseUserDataService to synchronize account settings and moderation data across devices. The service continues to manage user-specific data storage in SharedPreferences by prefixing keys with user IDs, maintaining complete data separation even on shared devices.

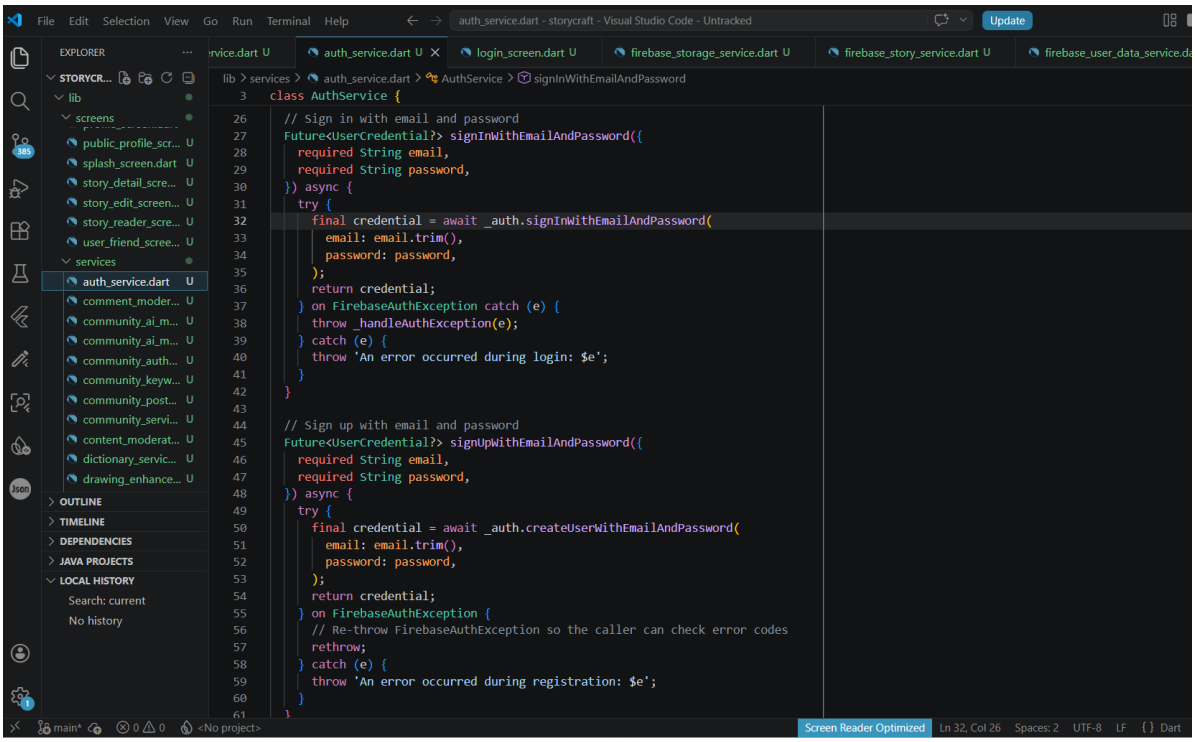


Figure 4.4.2.2 auth_service.dart

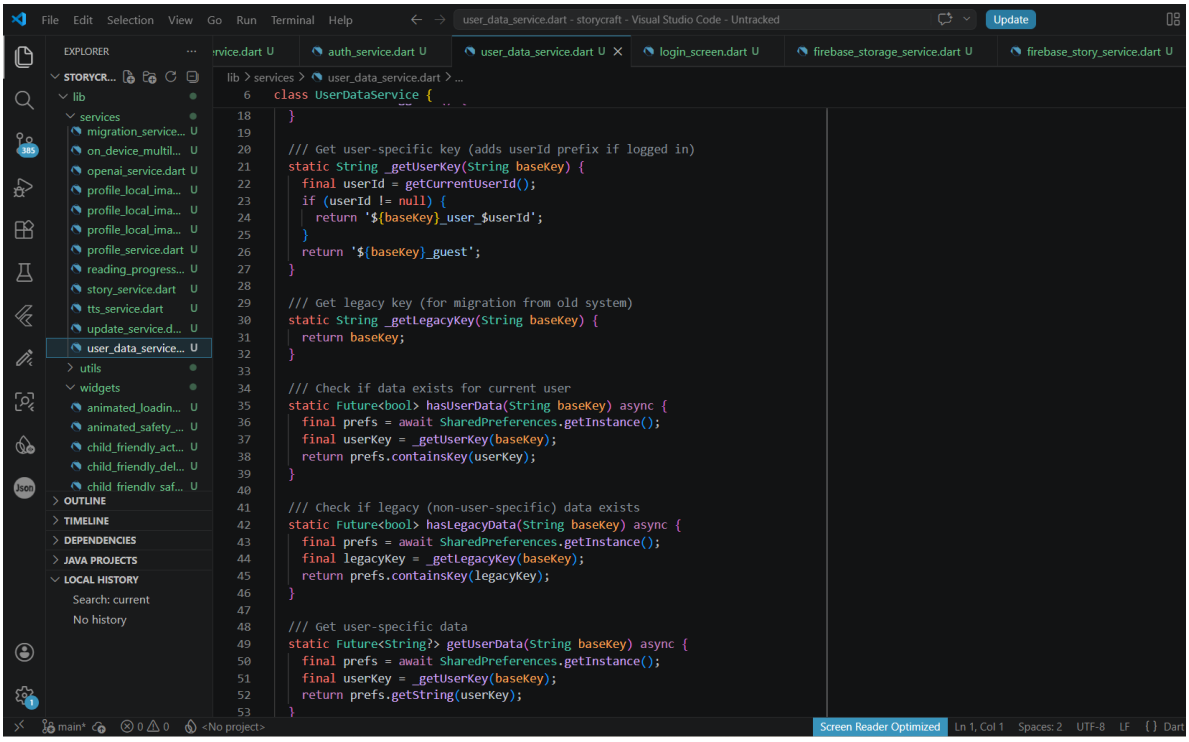


Figure 4.4.2.3 user_data_service.dart

Story service as shown in Figure 4.4.2.4 story_service.dart manages story data operations including loading stories from asset files and JSON sources, implementing caching mechanisms for efficient story retrieval that reduce load times. The service provides comprehensive search functionality that enables users to find stories by title, description, author, or content, along with filter operations that allow stories to be organized by various criteria.

Generated story service as shown in Figure 4.4.2.5 generated_story_service.dart handles user-generated stories that are created through the story generation process, storing these stories locally in the file system as JSON files. The service has been updated to manage the moderation lifecycle of each story, tracking whether an item is pending review, approved, or rejected by the administrative system. It implements complete CRUD for create, read, update, delete operations for story management, handles file system interactions for story storage and retrieval, and manages story versioning to support both original and enhanced versions of generated stories.

Reading progress service as shown in Figure 4.4.2.6 reading_progress_service.dart tracks user reading activities by storing the last page read for each story, enabling “continue reading” functionality that allows users to resume stories where they left off. The service also manages “Community Bookmarks”, allowing users to save posts from the community feed quickly into

their personal library for later reading. Progress synchronization with Firestore ensures that both reading progress and bookmarked community stories are accessible across multiple devices.

Profile service as shown in Figure 4.4.2.7 profile service.dart manages user profile information including username and profile picture management. The service is now responsible for monitoring user safety metrics, including strike count and community ban statuses. It ensures that usernames meet uniqueness requirements and handles the secure upload of profile avatars to Firebase Storage with proper metadata tagging.

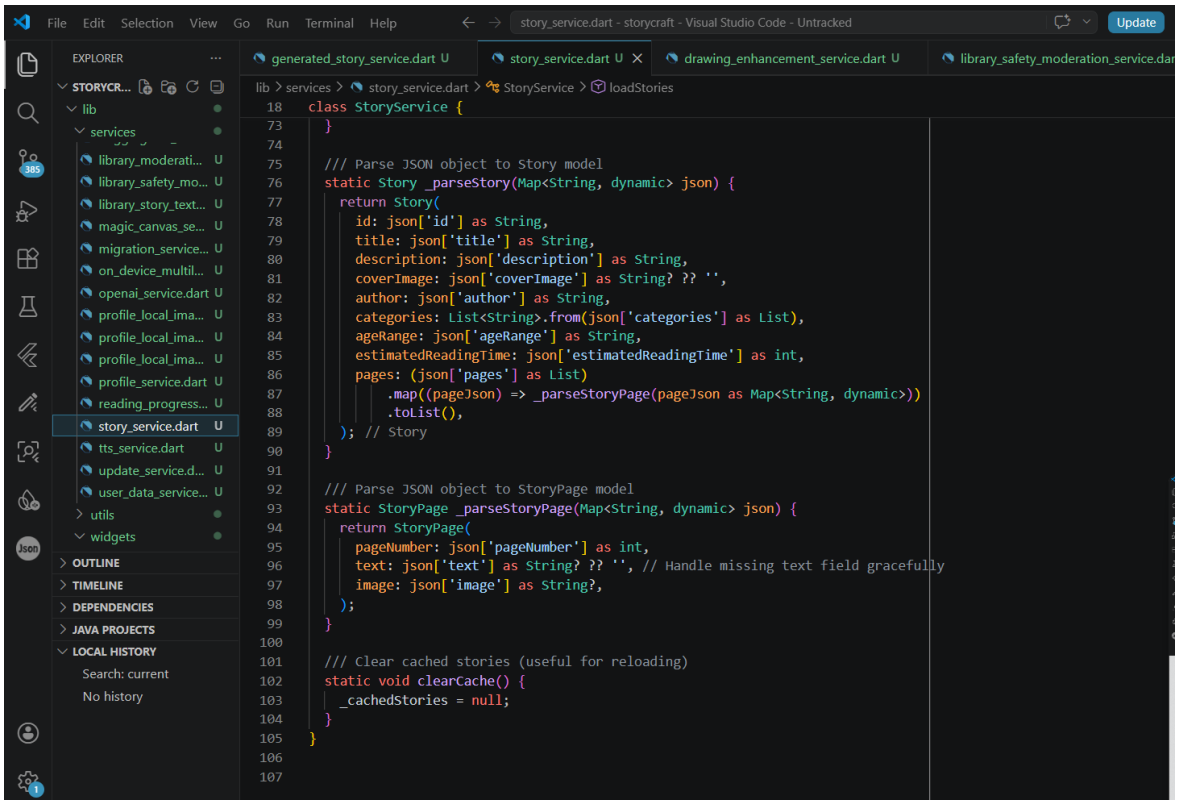


Figure 4.4.2.4 story_service.dart

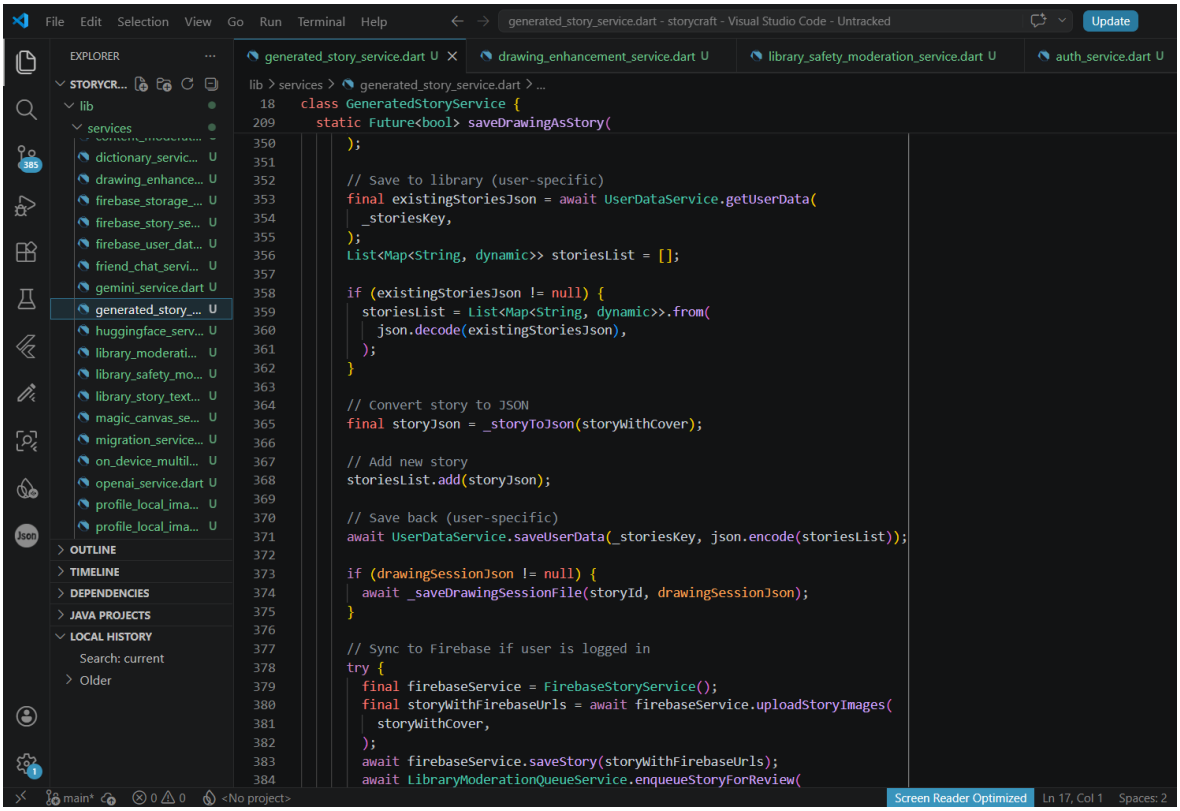


Figure 4.4.2.5 generated_story_service.dart

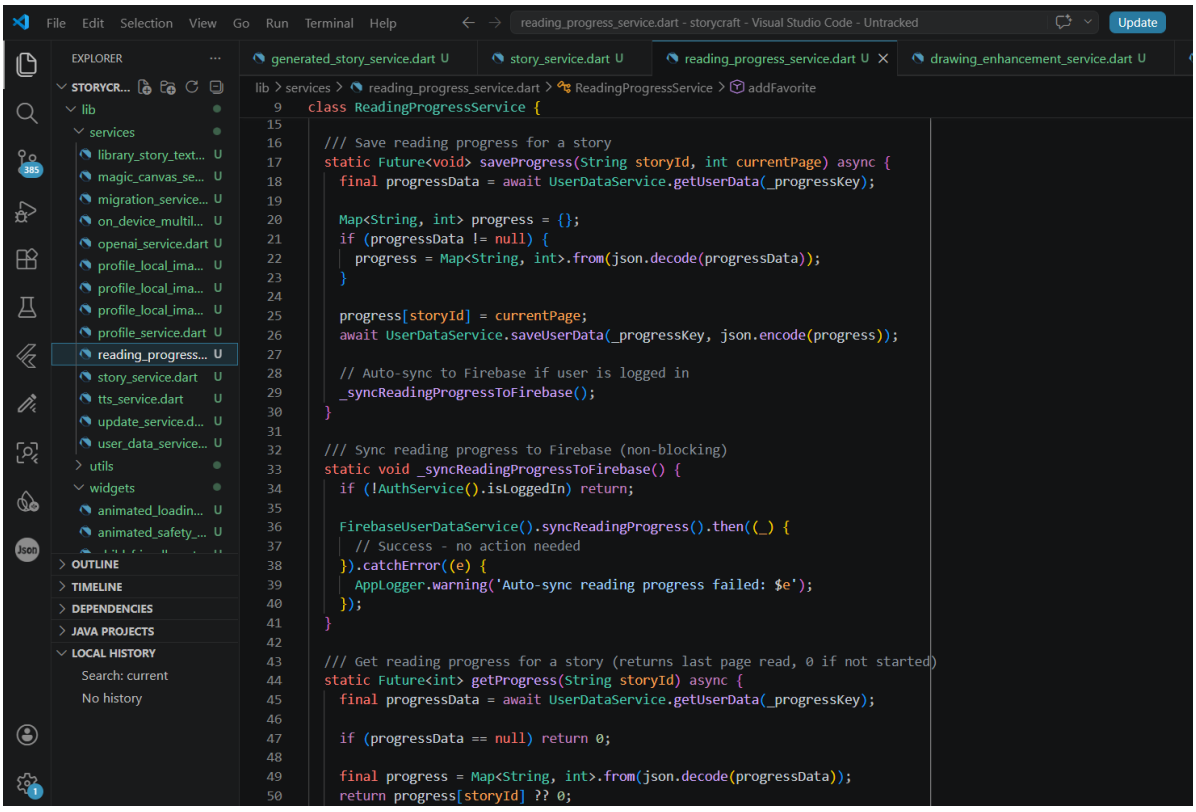


Figure 4.4.2.6 reading_progress_service.dart

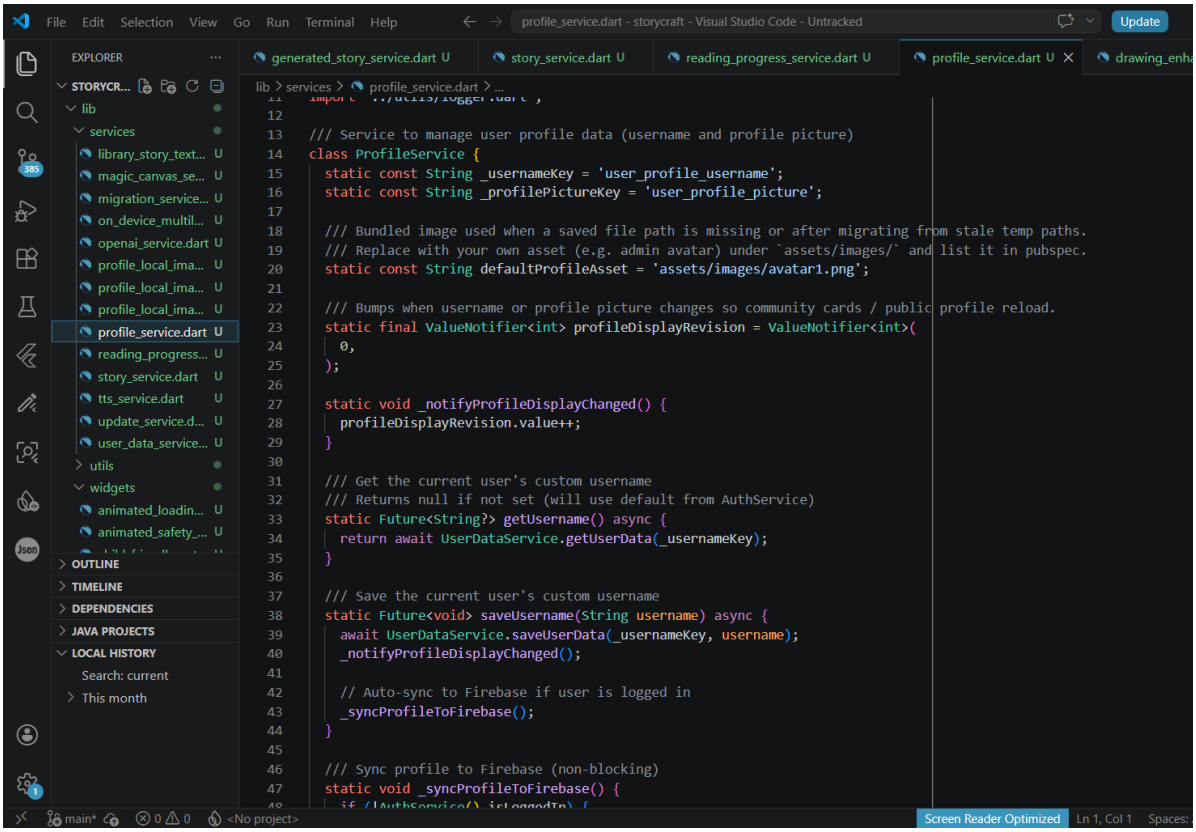


Figure 4.4.2.7 profile_service.dart

TTS service as shown in Figure 4.4.2.8 tts_service.dart initializes and manages text-to-speech functionality across multiple providers, handling voice provider selection that allows users to choose between system TTS, Google Cloud TTS, and Amazon Polly. Speech rate and pitch control enable customization of the narration experience, while word highlighting synchronization ensures that visual feedback matches audio playback precisely.

Drawing enhancement service as shown in Figure 4.4.2.9 drawing_enhancement_service.dart emphasizes image enhancement processes, coordinating with multiple AI providers including GPT-4 and DALL-E to improve user drawings. The service handles image processing and optimization to prepare images for AI analysis and manages both original and enhanced versions of drawings.

Gemini service as shown in Figure 4.4.2.10 gemini_service.dart provides dedicated integration with Google Gemini API. The service's role has expanded beyond story generation to include vision moderation, where it analyzes drawing for inappropriate content and provides detailed safety analysis reports for the administrative moderation queue. Response parsing and

formatting ensure that generated stories are properly structured and formatted for display in the application. It uses sophisticated prompt engineering to make sure all generated content remains child-friendly and genre consistent.

The application integrates with AI services using a multi-provider architecture to ensure high availability and content safety. While Google Gemini 1.5 Flash is the primary engine for both storytelling and real-time moderation, the OpenAI service as shown in Figure 4.4.2.11 openai_service.dart remains a critical component of the platform's reliability. It implements integration with OpenAI GPT-4 Vision API as a high-performance alternative story generation and enhancement provider. The service includes comprehensive error handling, automatic retry logic, and advanced response parsing that ensures children receive a seamless and coherent story experience even when the primary provider encounters connectivity or rate-limiting issues.

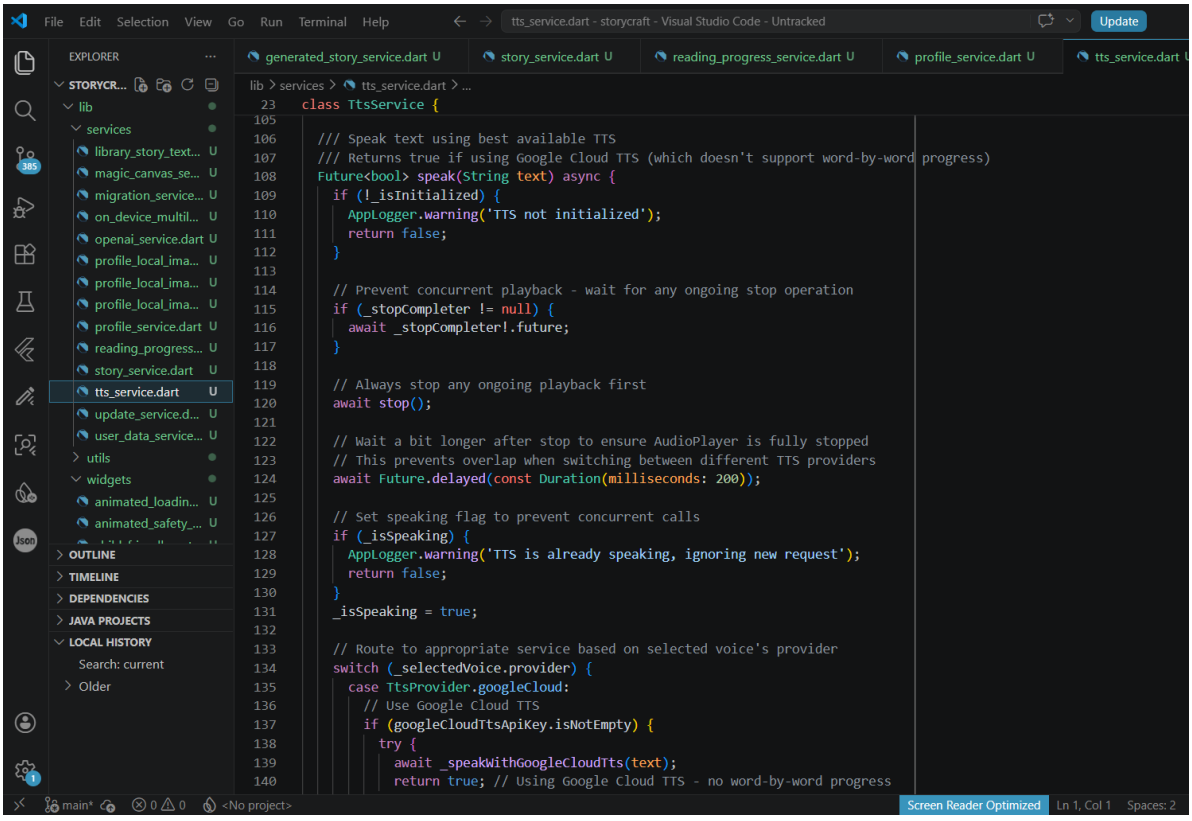


Figure 4.4.2.8 tts_service.dart

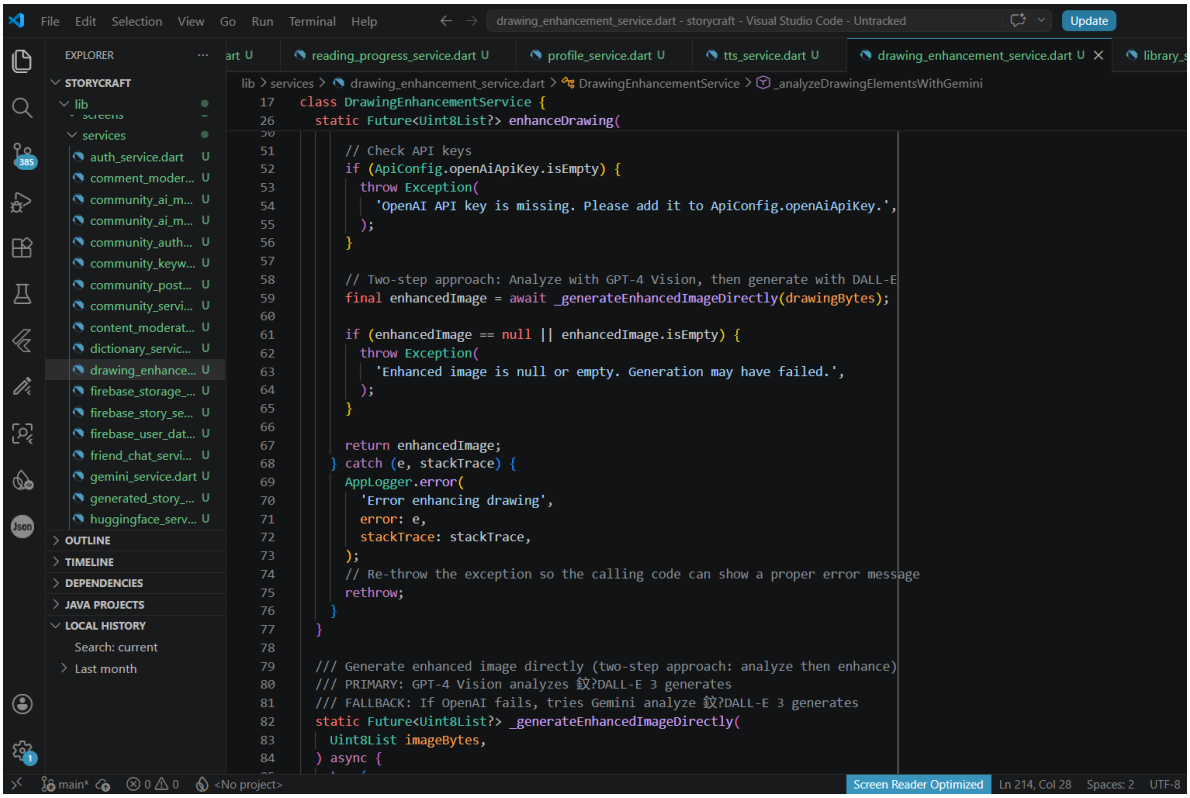


Figure 4.4.2.9 drawing_enhancement_service.dart

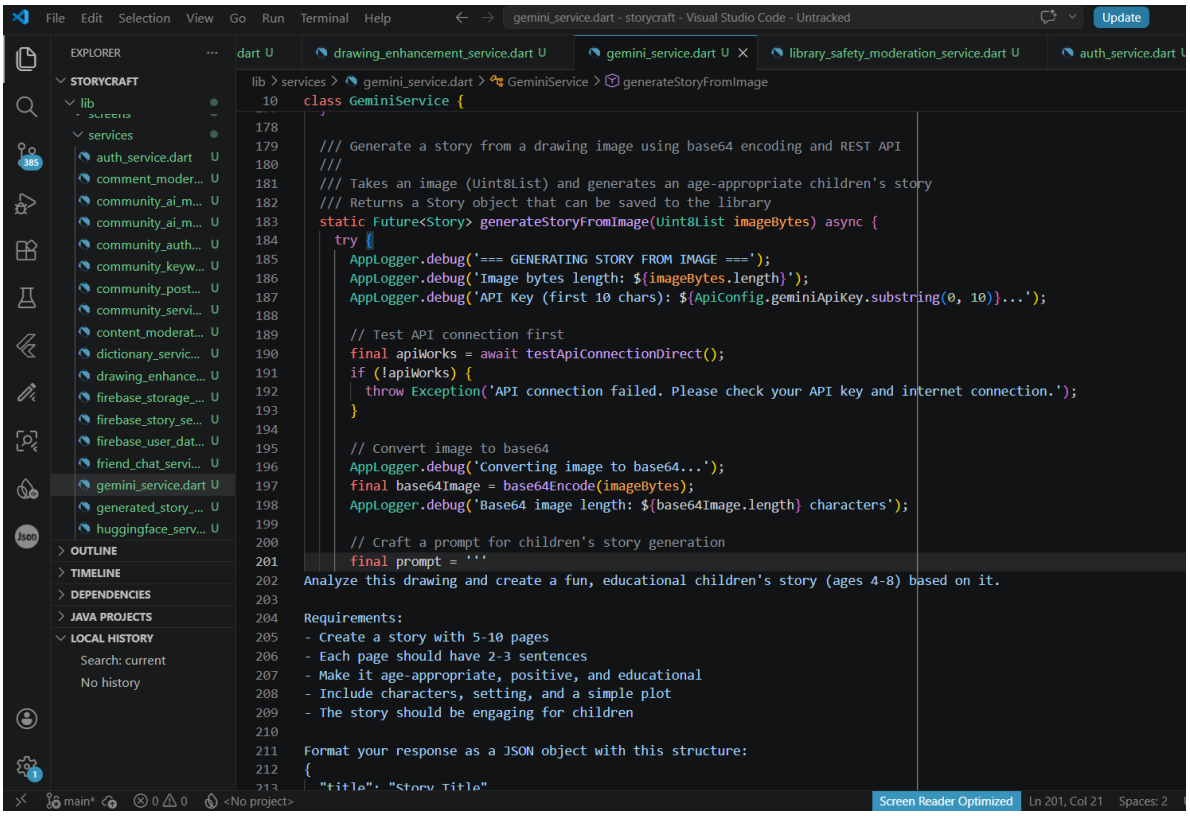


Figure 4.4.2.10 gemini_service.dart

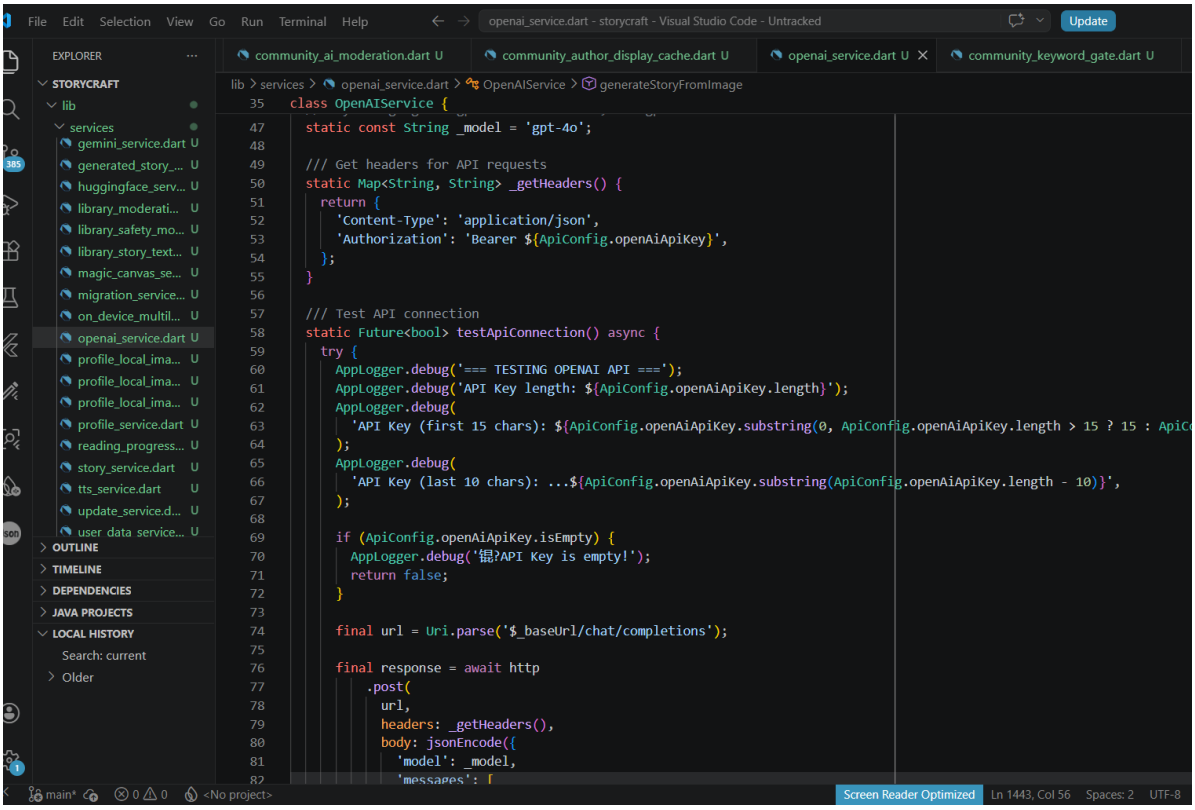


Figure 4.4.2.11 openai_service.dart

Community services as shown in Figure 4.4.2.12 community_service.dart is the central service that manages all social interactions within the platform. It handles the submission and retrieval of community posts, manages the like and unlike operations, processes user reports on inappropriate content, and enforces posting restrictions for banned accounts. The service communicates directly with Cloud Firestore to synchronize community data in real time across all users.

Community AI moderation service as shown in Figure 4.4.2.13 for the community_ai_moderation_service.dart implements a real-time safety filter specifically for social interactions. The primary role of this service is to moderate community comments; it utilizes Gemini 1.5 Flash to contextually analyze every user comment before it is published. This ensures that the platform can detect and block inappropriate content that simple keyword filters might miss, such as bullying, hidden insults or leetspeak. By processing comments through this AI-driven service, the application maintains a respectful and safe environment for

its young audience, allowing for real-time social engagement while preventing toxic interactions from becoming visible in the community feed.

Library moderation queue service as shown in Figure 4.4.2.14 for the library_moderation_queue_service.dart manages the queue of drawings and stories submitted to the library for administrative review. It retrieves pending, approved, and rejected items from Firestore, and provides methods for administrative to approve or reject submissions. Upon rejection, the service automatically increments the author's strike count and dispatches a notification to inform the user of the moderation outcome.

Friend chat service as shown in Figure 4.4.2.15 friend_chat_service.dart manages the real-time private messaging functionality between community members. It maintains conservation streams, handles message persistence to Cloud Firestore, tracks unread message counts per conservation, and provides the data necessary to display the latest message preview in the inbox. The service ensures that message updates are reflected instantly for both the sender and recipient.

Dictionary service as shown in Figure 4.4.2.16 dictionary_service.dart provides word definition lookup functionality to support the educational objectives of the application. It communicates with an external dictionary API called free dictionary API to fetch definitions, phonetic transcriptions, and example sentences for works selected by the user during reading. The service formats the API response into a simplified structure suitable for child-friendly display.

The application implements a multi-stage comment filtering system to ensure a respectful and safe environment for children. Every comment submission is routed through the comment moderation service before being saved to the database as shown in Figure 4.4.2.17 community_moderation_service.dart. This service utilizes the community keyword gate for immediate local screening of restricted terms, followed by a contextual analysis using the community AI moderation service. This AI layer is specifically tuned to detect and block sophisticated bypass attempts, such as obfuscated profanity, leetspeak, and implicit bullying, ensuring that only genuinely kind and safe interactions are published to the community feed.

The application maintains a consistent safety standard across both the private library and public community by utilizing a unified scoring logic. As shown in Figure 4.4.2.18 library_safety_moderation_service.dart, the library service employs 0-100 risk scale and

threshold tiers (<20% for auto-approval, 20-49% for admin attention, and ≥50% for high-risk flagging). The library moderation is specifically tailored for visual content, evaluating not only violence and sexual themes but also child-specific risks such as bullying, frightening graphic content, and other unsafe themes. This ensures that even before a story is shared with the community, it has already passed a rigorous multi-dimensional safety screening within the user's private library, providing an extra layer of protection for the young audience.

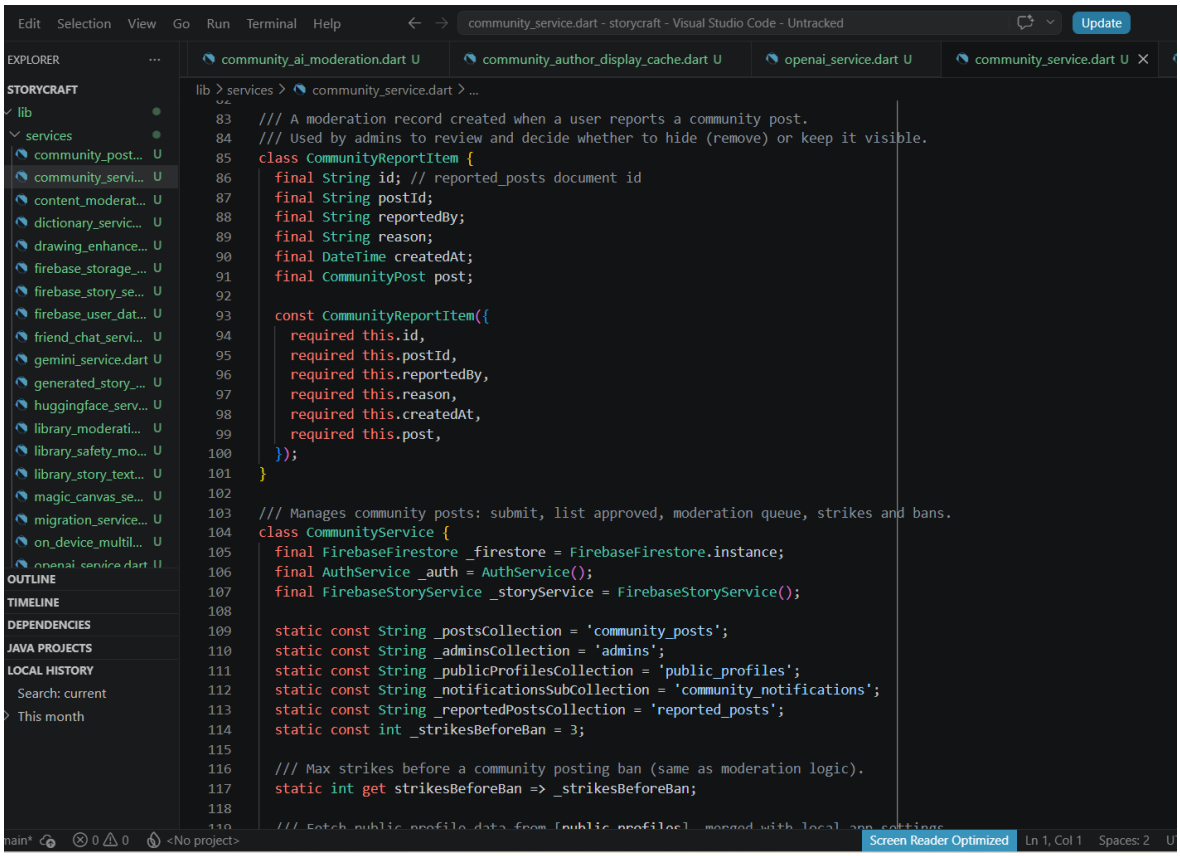
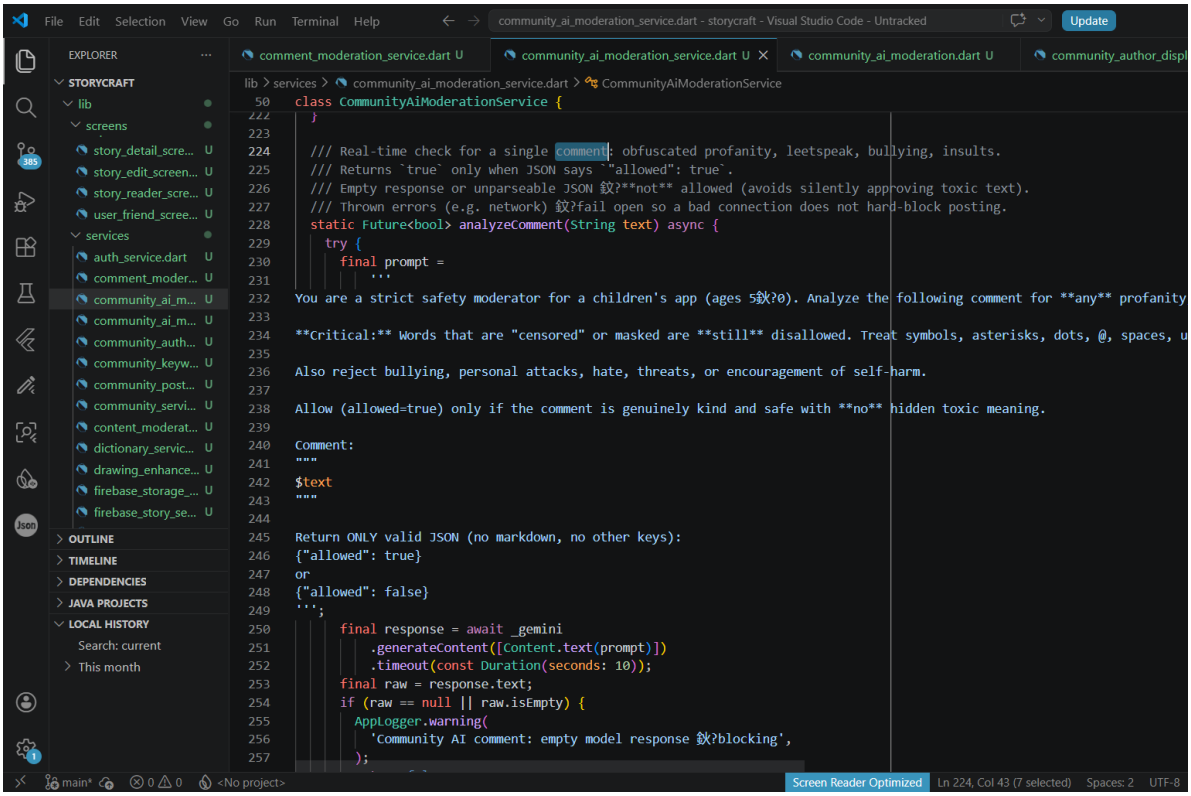
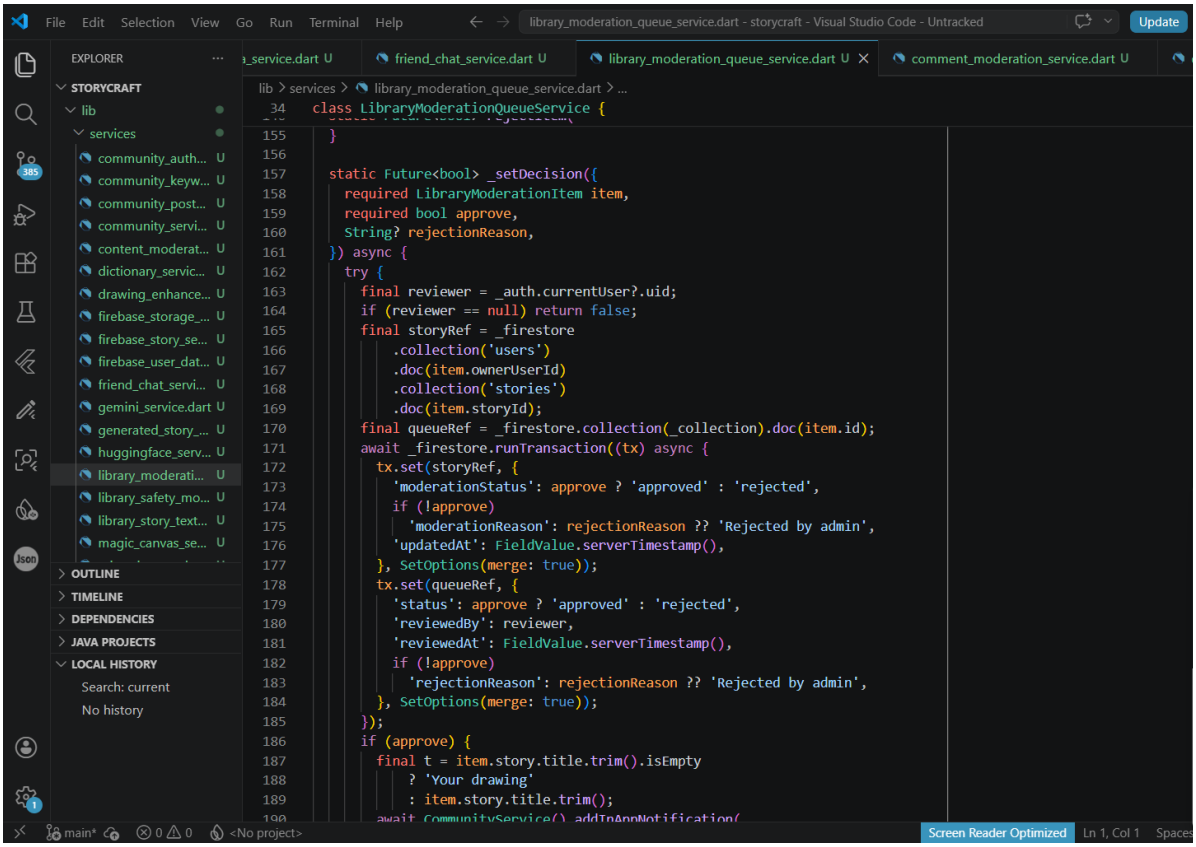


Figure 4.4.2.12 community_service.dart



```dart
class CommunityAiModerationService {
  }

  /// Real-time check for a single comment: obfuscated profanity, leetspeak, bullying, insults.
  /// Returns `true` only when JSON says `"allowed": true`.
  /// Empty response or unparseable JSON 鈥?**not** allowed (avoids silently approving toxic text).
  /// Thrown errors (e.g. network) 鈥?fail open so a bad connection does not hard-block posting.
  static Future<bool> analyzeComment(String text) async {
    try {
      final prompt =
          '''
You are a strict safety moderator for a children's app (ages 5鈥?0). Analyze the following comment for **any** profanity

**Critical:** Words that are "censored" or masked are **still** disallowed. Treat symbols, asterisks, dots, @, spaces, u

Also reject bullying, personal attacks, hate, threats, or encouragement of self-harm.

Allow (allowed=true) only if the comment is genuinely kind and safe with **no** hidden toxic meaning.

Comment:
"""
$text
"""

Return ONLY valid JSON (no markdown, no other keys):
{"allowed": true}
or
{"allowed": false}
''';
      final response = await _gemini
          .generateContent([Content.text(prompt)])
          .timeout(const Duration(seconds: 10));
      final raw = response.text;
      if (raw == null || raw.isEmpty) {
        AppLogger.warning(
          'Community AI comment: empty model response 鈥?blocking',
        );
```

Figure 4.4.2.13 community_ai_moderation_service.dart



```dart
class LibraryModerationQueueService {
  }

  static Future<bool> _setDecision({
    required LibraryModerationItem item,
    required bool approve,
    String? rejectionReason,
  }) async {
    try {
      final reviewer = _auth.currentUser?.uid;
      if (reviewer == null) return false;
      final storyRef = _firestore
          .collection('users')
          .doc(item.ownerUserId)
          .collection('stories')
          .doc(item.storyId);
      final queueRef = _firestore.collection(_collection).doc(item.id);
      await _firestore.runTransaction((tx) async {
        tx.set(storyRef, {
          'moderationStatus': approve ? 'approved' : 'rejected',
          if (!approve)
            'moderationReason': rejectionReason ?? 'Rejected by admin',
          'updatedAt': FieldValue.serverTimestamp(),
        }, SetOptions(merge: true));
        tx.set(queueRef, {
          'status': approve ? 'approved' : 'rejected',
          'reviewedBy': reviewer,
          'reviewedAt': FieldValue.serverTimestamp(),
          if (!approve)
            'rejectionReason': rejectionReason ?? 'Rejected by admin',
        }, SetOptions(merge: true));
      });
      if (approve) {
        final t = item.story.title.trim().isEmpty
            ? 'Your drawing'
            : item.story.title.trim();
```

Figure 4.4.2.14 library_moderation_queue_service.dart

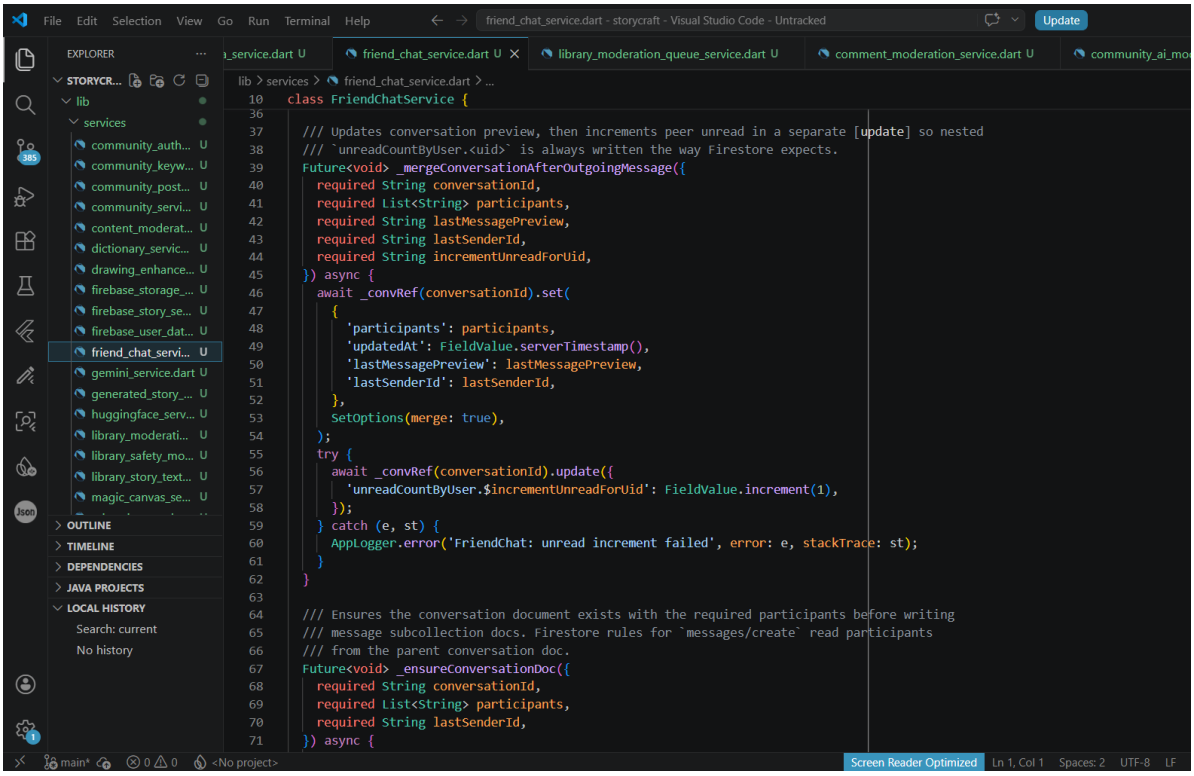


Figure 4.4.2.15 friend_chat_service.dart

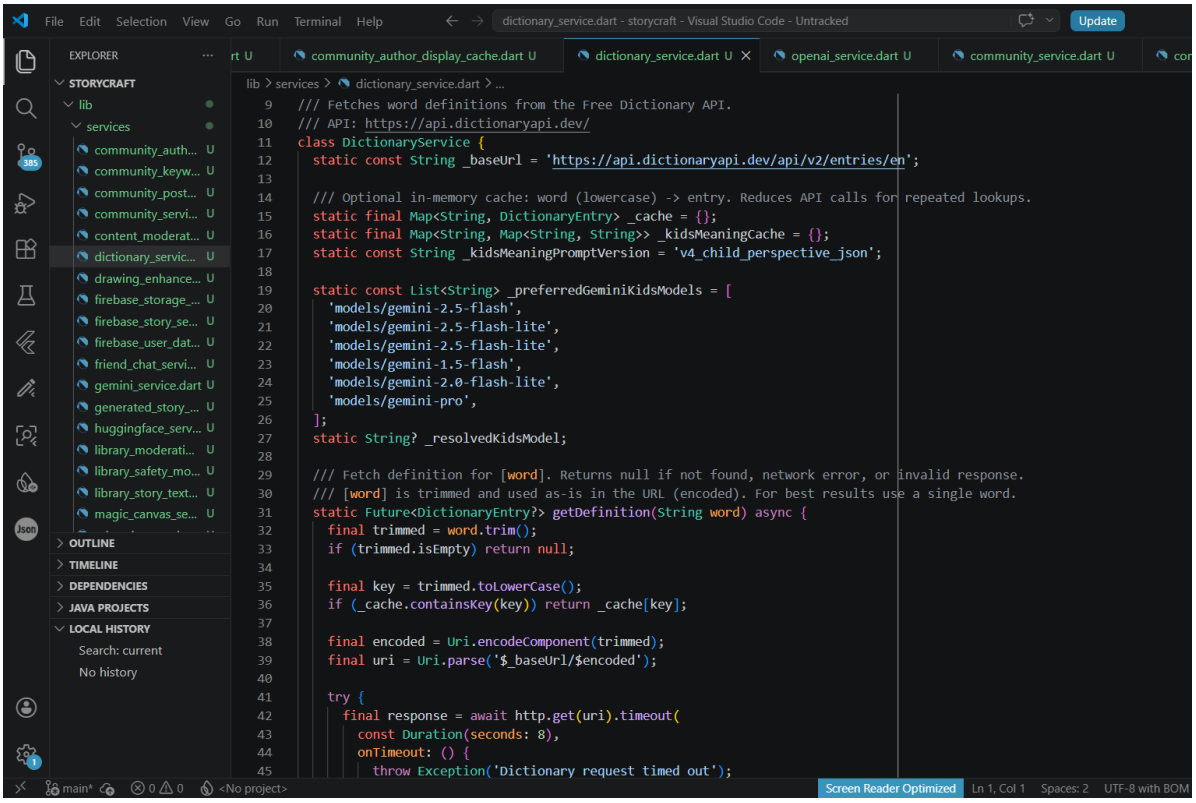


Figure 4.4.2.16 dictionary_service.dart

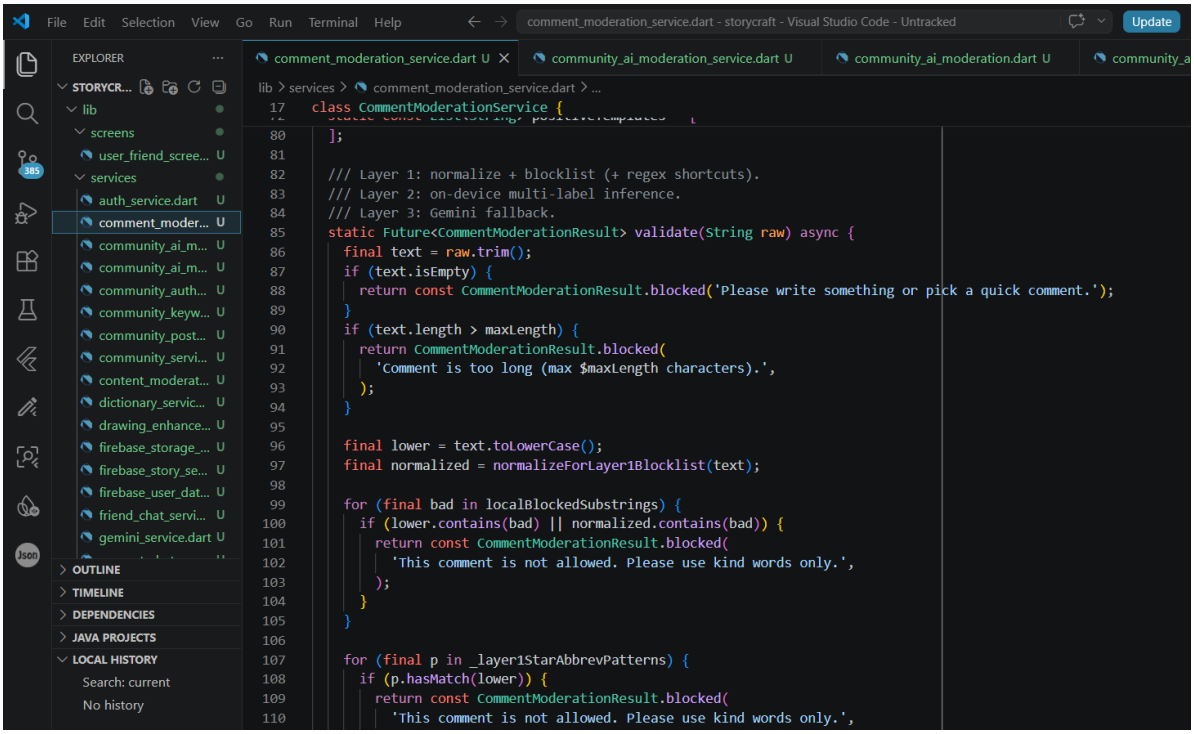


Figure 4.4.2.17 community_moderation_service.dart

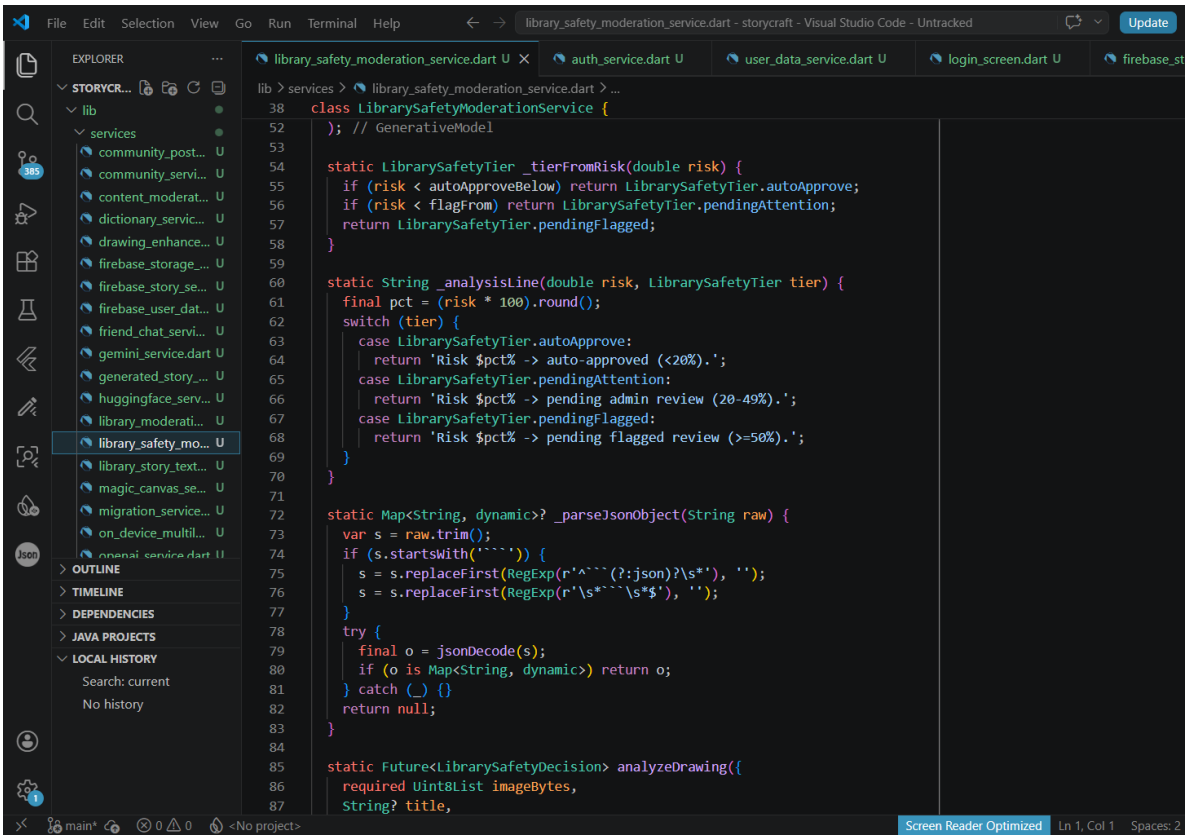


Figure 4.4.2.18 library_safety_moderation_service.dart

4.4.3 System Implementation and Deployment

To compile and run the StoryCraft application, the development environment requires Flutter SDK and Dart SDK. The backend is initialized by linking the application to a Firebase Project using the flutterfire configure comman, which generates the firebase_options.dart file. To deploy the system to a physical Android device and emulator, the following steps are performed:

- Enable Developer Options and USB Debugging on the target device.
- Configure the local.properties file with the correct Android SDK path.
- Execute flutter pub get to install all dependencies.
- Run the flutter run --release to compile the Dart code into a native APK using the AOT (Ahead-of-Time) compiler and upload it to the device.

4.5 Summary

This chapter provided a comprehensive overview of the technical development and architectural implementation of the StoryCraft application. The frontend development utilized the Flutter framework to create a responsive, cross-platform experience optimized for children, while the backend leveraged a Backend-as-a-Service (BaaS) model through Firebase to handle real-time data and authentication. By detailing the Model-View-Service (MVS) architecture, the NoSQL database schema, and the security frameworks, this chapter has shown how the system was engineered to be scalable, secure and user-centric. These development foundations directly support the successful achievement of the project's objectives, which will be further analyzed in the subsequent testing and results chapters.

CHAPTER 5 IMPLEMENTATION AND TESTING

5.1 Overview

This chapter presents the finalized implementation and testing of the StoryCraft application. It provides a visual walkthrough of the complete user interface, highlighting the integration of AI story generation with the social community and moderation features. The chapter is organized into functional modules, ranging from main creativity tools to administrative safety dashboards. Furthermore, it details the testing methodology, which focuses on functional validation and safety verification to ensure a secure experience for children. By presenting both the finalized screens and verification results, this chapter demonstrates that StoryCraft is a safe, stable, and fully functional platform that meets all project requirements.

5.2 Splash Screen

The splash screen in Figure 5.2.1 serves as the initial entry point of the StoryCraft application, presenting users with the app's branding and logo immediately upon launch. This screen plays a critical role in the app initialization process, as it handles essential background tasks that must be completed before users can interact with the application. During this brief loading period, the system establishes a connection with Firebase services, initializes core application services such as the UpdateService for version checking, and determines the appropriate navigation path based on the user's authentication status. The screen features a clean and minimal design, displaying the StoryCraft logo prominently in the center, accompanied by a loading indicator that provides visual feedback that the application is initializing. The splash screen automatically transitions to either the login screen if the user is not authenticated, or directly to the main navigation screen if the user has an active session. This automatic navigation ensures a seamless user experience by eliminating the need for manual interaction during the app startup sequence.

Figure 5.2.1 Splash Screen

5.3 Login Screen

The login screen in Figure 5.3.1 serves as the primary authentication gateway for the StoryCraft application, providing users with a straightforward interface for accessing the app's features through email and password authentication. This screen accommodates two primary user scenarios, such as returning users who wish to log in with their existing credentials, and new users who need to create an account. The interface is designed with simplicity and clarity in mind, featuring clearly labeled email and password input fields that guide users through the authentication process. The screen offers standard email and password login for registered users, with a toggle option that allows users to switch between login and sign-up modes seamlessly. Users can easily switch from login to sign up mode by tapping the "Don't have an account? Sign Up" link at the bottom of the screen, which transforms the interface to display the registration form with additional fields for password confirmation. When users successfully authenticate, they are automatically navigated to the main navigation screen, while authentication errors are displayed through user-friendly error messages that appear below the input fields, allowing users to correct their input and retry the authentication process as shown

in Figure 5.3.2. The screen implements comprehensive form validation that provides real-time feedback on email format and password requirements, ensuring users enter valid credentials before attempting authentication.

Figure 5.3.1 Login Screen

Figure 5.3.2 Login Error Screen

5.4 Sign Up Screen

The sign up screen as shown in Figure 5.4.1 is accessed by toggling from the login screen and provides a dedicated interface for new users to create their StoryCraft accounts. The screen features a distinct visual design with a lighter purple gradient background that differentiates it from the login interface, displaying a prominent “Create Account” title at the top with a back arrow button that allows users to return to the login view. A significant addition in this version is the username field, which is essential for the application’s new community and social features. The interface includes real-time username validation that ensures each identifier is unique and follows specific formatting rules, such as a minimum of three characters and the user of only alphanumeric characters, underscores, or dots as shown in Figure 5.4.2. The screen presents a user-friendly registration form with individual card-based input fields for username, email, password, and password confirmation. The email field includes validation for proper

email format, while the password field features real-time password strength requirements that appear as users type, showing criteria such as minimum length, uppercase and lowercase letters, and numeric characters as shown in Figure 5.4.3. The confirm password field includes visual feedback to indicate when passwords match as shown in Figure 5.4.4. Upon successful registration, a success message is displayed as shown in Figure 5.4.5 and automatically switches back to login mode after a brief delay, prompting users to log in with their newly created credentials. The interface includes a toggle option at the bottom that allows users to switch back to login mode if they already have an account. If the user signs up with an email or username that is already registered, it will show “You are already a member of StoryCraft. Please log in to access.” as shown in Figure 5.4.6. If the user enters the username that is already taken by another user, then it will show “This username is already taken. Please choose another one.” as shown in Figure 5.4.7.

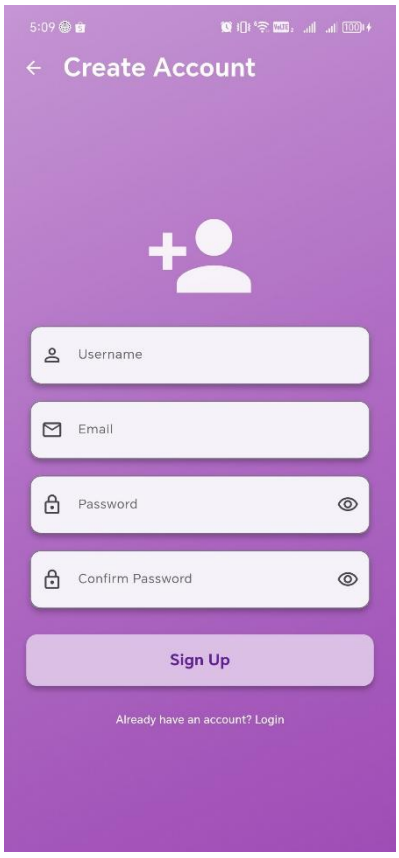


Figure 5.4.1 Sign Up Screen

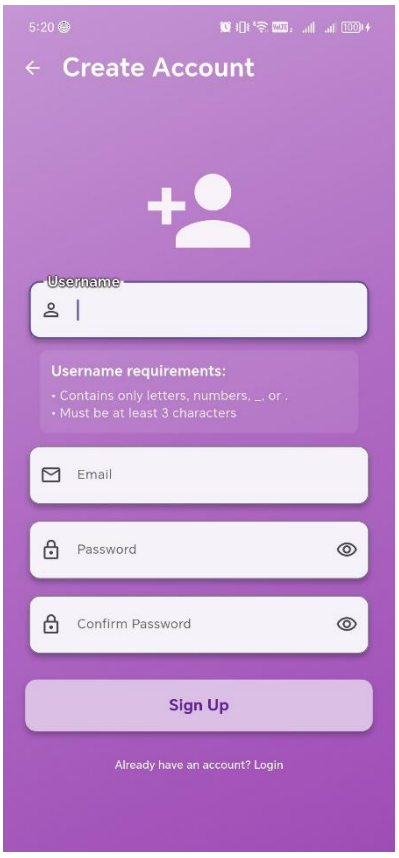


Figure 5.4.2 Username Requirement

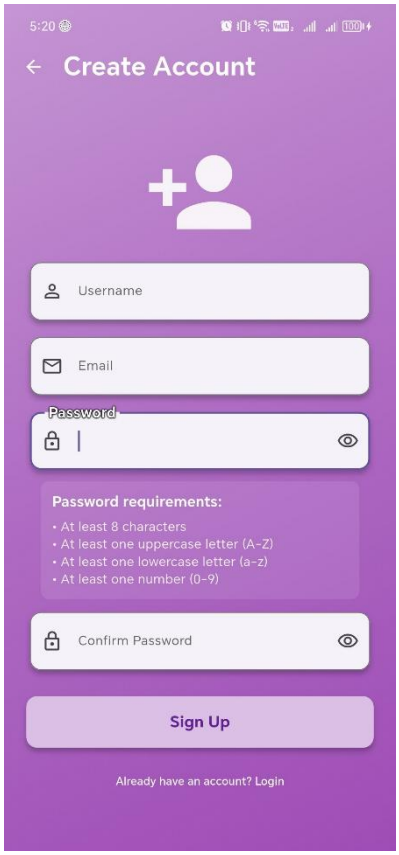


Figure 5.4.3 Password Format

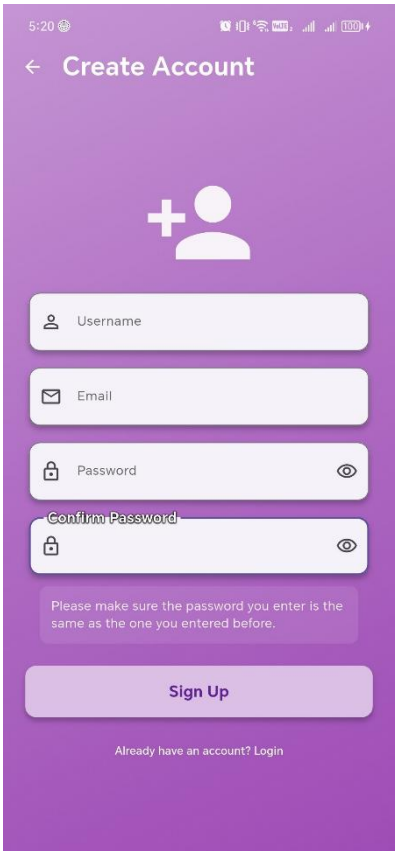


Figure 5.4.4 Confirm Password

Figure 5.4.5 Account Creation

Figure 5.4.6 Registered Account

Figure 5.4.7 Unique Username

5.5 Home Screen

The home screen shown in Figure 5.5.1 serves as the primary landing page and central hub of the StoryCraft application, providing users with an overview of all available regular stories immediately after login. This screen is strategically designed to engage users from the moment they enter the application, displaying a “Discover Stories” section that presents all available regular stories in a scrollable list format. The screen filters out user-generated drawings and generated stories, focusing exclusively on the predefined story collection. A welcome dialog shown in Figure 5.5.2 appears for new users, providing an introduction to the app’s features and guiding them through their initial experience. The app bar at the top of the screen includes a search icon that toggles a search bar, allowing users to search stories by title, author, or category with real-time filtering as they type as shown in Figure 5.5.3. The screen also features a “Surprise me” button in the app bar as shown in Figure 5.5.4 that randomly selects a story from the available collection, providing an engaging way for users to discover new content. The random story screen is as shown in Figure 5.5.5 after the user click “Surprise Me” button and the user can select to pick another story by clicking the “Pick

Another" button or "Read Now" to read it. Additionally, the screen includes category filtering functionality as shown in Figure 5.5.6 that allows users to filter stories by specific categories, with a horizontally scrollable category selector that displays available story categories including an "All" option to view all stories. The screen's layout emphasizes visual storytelling through story cards that display cover images, titles, and brief descriptions, encouraging users to explore the available content. The screen supports pull-to-refresh functionality that reloads the story list and provides helpful prompts about the random pick feature.

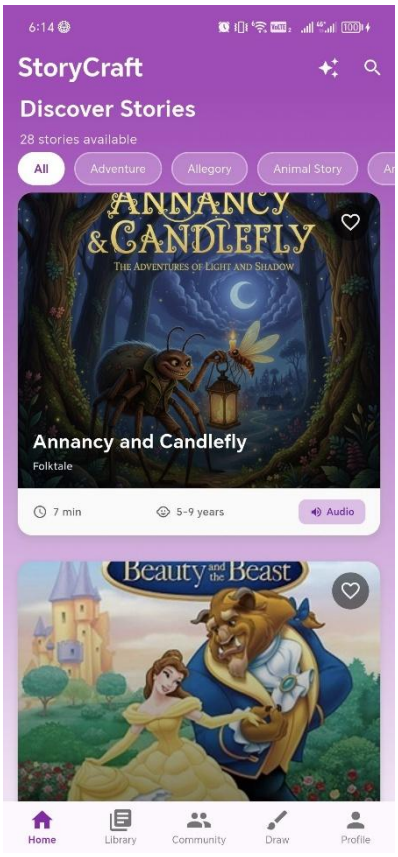


Figure 5.5.1 Home Screen

Figure 5.5.2 Welcome Message Dialog

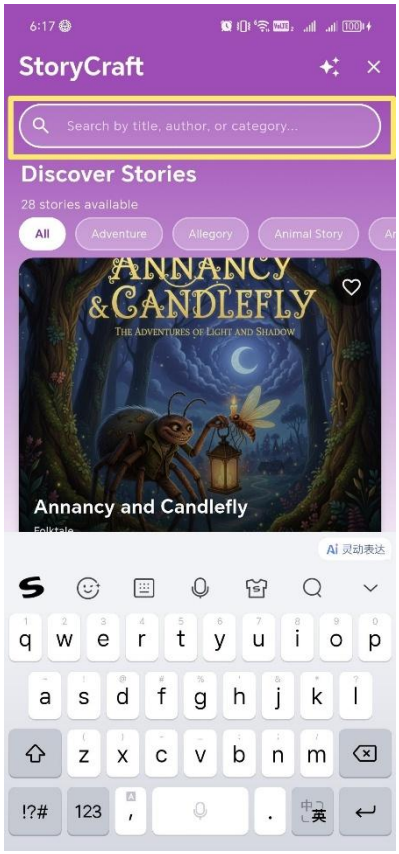


Figure 5.5.3 Home Search

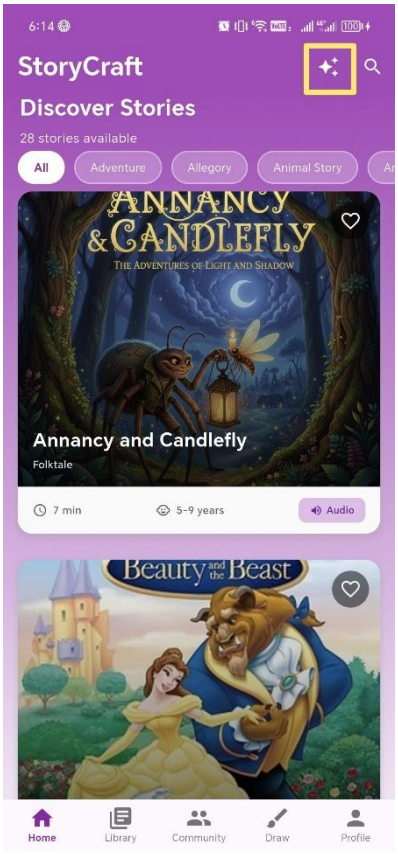


Figure 5.5.4 Random Pick Story

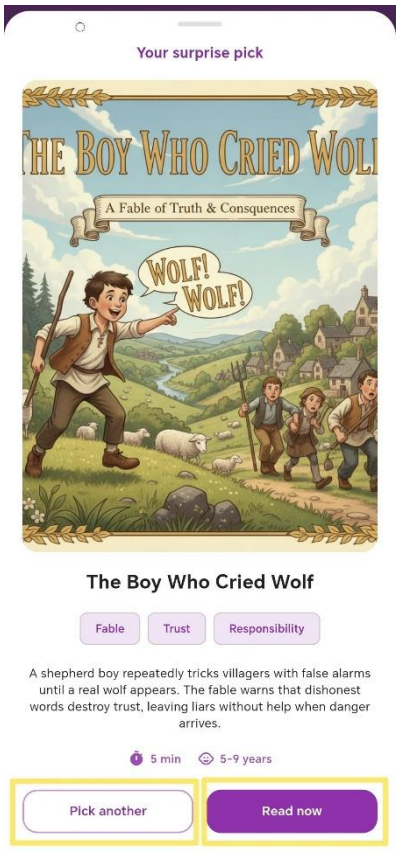


Figure 5.5.5 Random Story Screen

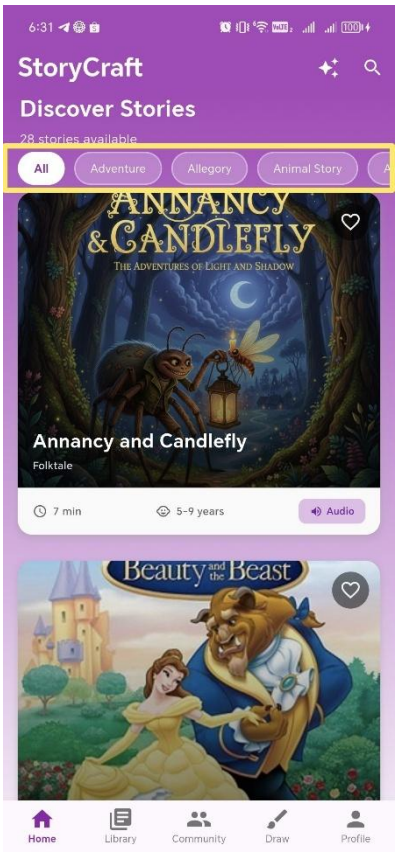


Figure 5.5.6 Home Filter

5.6 Library Screen

The library screen functions as the comprehensive content management center of the StoryCraft application, providing users with complete access to their entire story collection through an organized and searchable interface as shown in Figure 5.6.1. This screen displays stories in an intuitive grid or list view layout, with each story represented by a visually appealing card that showcases the cover image, title, and relevant metadata. The library screen is designed to accommodate users with extensive story collections, implementing efficient rendering techniques that ensure smooth scrolling and quick access to any story in the user's library. The screen features an integrated search bar at the top as shown in Figure 5.6.2 that enables real-time searching across story titles, authors, and categories, providing users with instant results as they type their search queries. Additionally, the library screen offers comprehensive filtering capabilities through a series of filter options as shown in Figure 5.6.3 that allow users to organize their stories by various criteria, including all stories, continue reading, favorites, generated stories, my drawings, and saved (community). Each story card displays visual indicators such as progress bars for partially read stories and favorite icons for marked stories, providing users with immediate visual cues about their reading status and preferences. The screen supports pull-to-refresh functionality as shown in Figure 5.6.4 to update the story list with the latest content, and users can tap on any story card to navigate to the story detail screen for more information and reading options. All the user-generated drawing and stories saved to library already pass through the safety check to ensure no violence or sexual content before share to community. The unsafe drawings and stories that might contain unsafe contents need to pass through admin review to perform actions like approve or reject before saving it to library to ensure a safety environment in library screen and community screen.

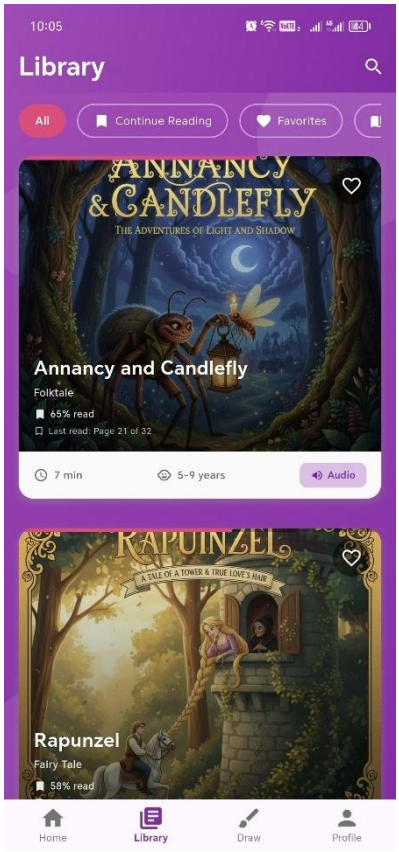


Figure 5.6.1 Library Screen

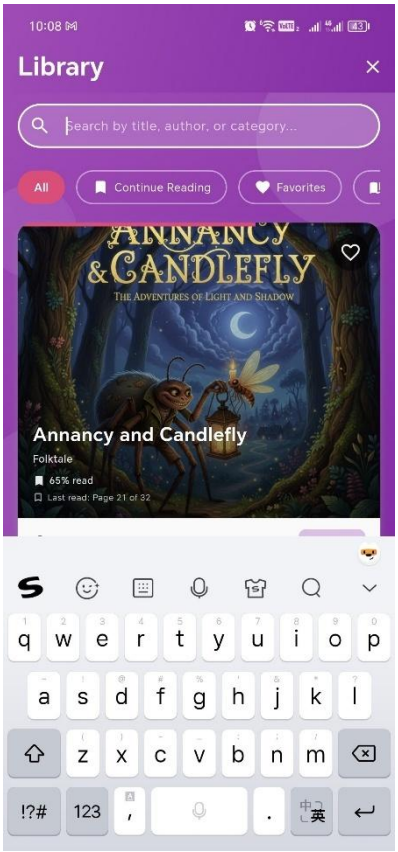


Figure 5.6.2 Library Search

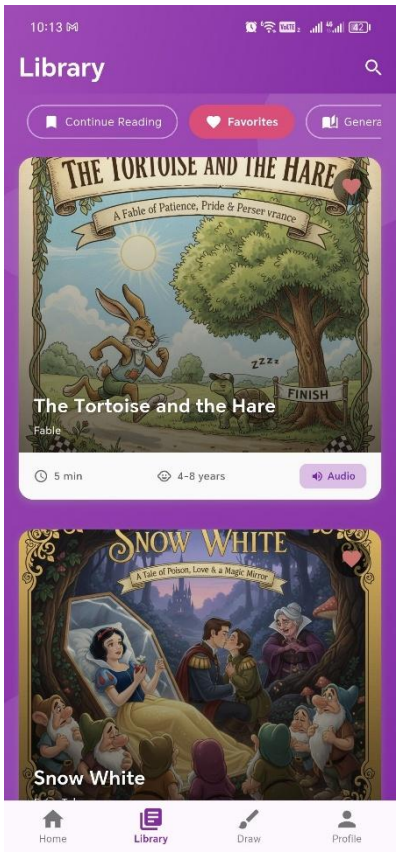


Figure 5.6.3 Library Filter

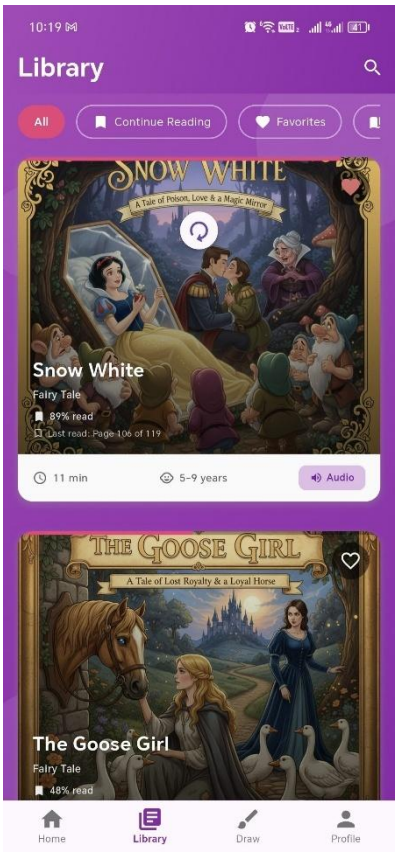


Figure 5.6.4 Library Refresh

5.7 Drawing Screen

The drawing screen as shown in Figure 5.7.1 provides users with a friendly reminder to ensure their drawing is always safe for everyone like did not draw any violence or sexually related drawing. The screen shows a comprehensive digital art creation environment, featuring a responsive touch-sensitive canvas that enables freehand drawing using a finger or stylus. This screen serves as the creative foundation of the StoryCraft application, allowing users to express their artistic vision before transforming their artwork into personalized stories through story generation service. The drawing interface is thoughtfully designed to balance functionality with ease of use, providing a rich set of drawing tools while maintaining an intuitive user experience suitable for users of all ages. The screen features a full-screen canvas that utilizes Flutter's CustomPaint widget for high-performance rendering, ensuring smooth and responsive drawing experiences even with complex artwork. A comprehensive toolbar provides access to various drawing tools including brushes for freehand drawing, shape tools for creating geometric forms, and an eraser for corrections. Users can customize their drawing experience as shown in Figure 5.7.2 through an integrated color picker that offers a wide spectrum of colors, and a brush size adjustment slider that allows for precise control over stroke width. The screen implements undo and redo functionality, enabling users to experiment freely without fear of making mistakes as shown in Figure 5.7.3. Once users have completed their drawing, they can access an options menu that presents several actions including drawing enhancement, story generation from the drawing, saving the artwork to the device gallery, or clearing the canvas to start fresh. When the users choose to save or improve their drawing, the application will let the users enter the drawing title as they want as shown in Figure 5.7.4. The users are also allowed to use their own image from devices gallery to generate a story for them as shown in Figure 5.7.5. The drawing is captured as a high-resolution image that is optimized for processing while maintaining visual quality. After the users perform either save or improve drawing actions from drawing screen, the application will automatically perform real-time safety check on the user’s drawing using Gemini Vision API to ensure it contains no inappropriate imagery before saving it to library. If contain any unsafe content for drawing, then the application will show a pending review reminder for user. If the admin approve on that drawing, then the drawings will automatically save into library for further editing.

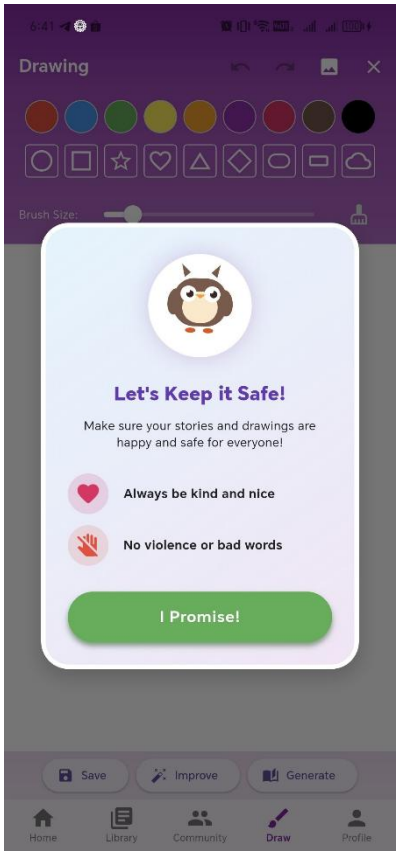


Figure 5.7.1 Safety Draw Dialog

Figure 5.7.2 Draw and Tools Screen

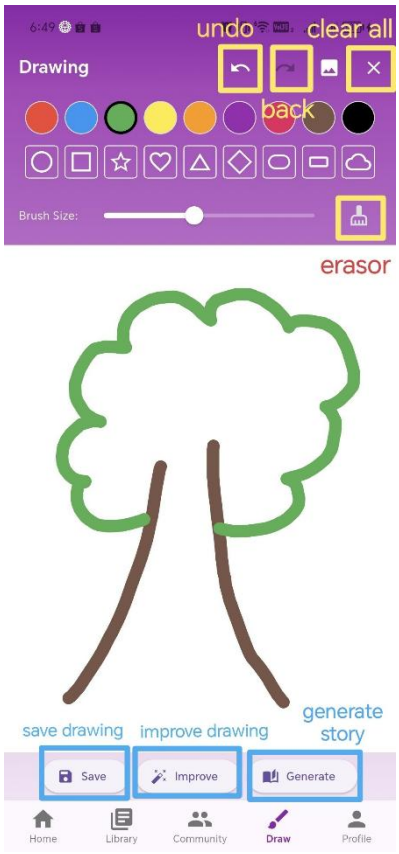


Figure 5.7.3 Draw Screen Button

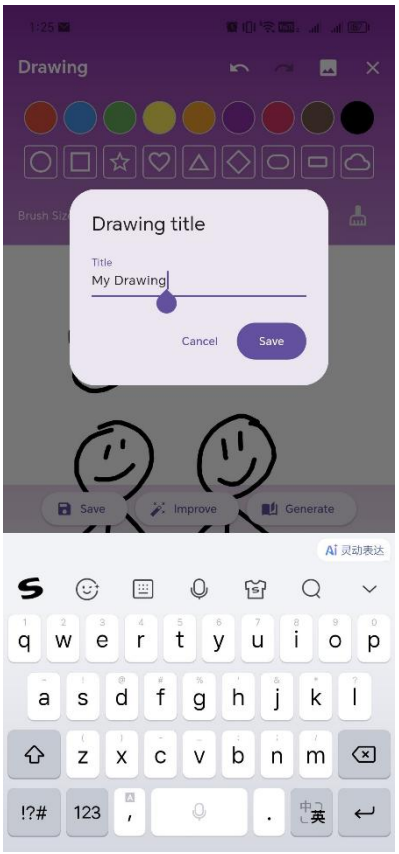


Figure 5.7.4 Drawing Title Input

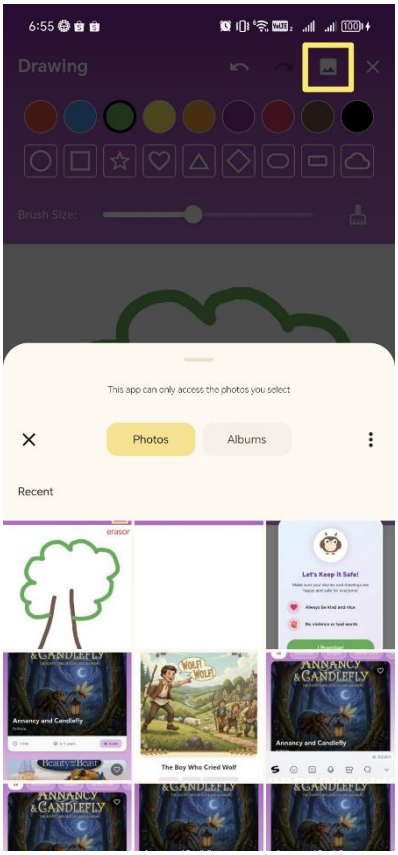


Figure 5.7.5 Import from Gallery

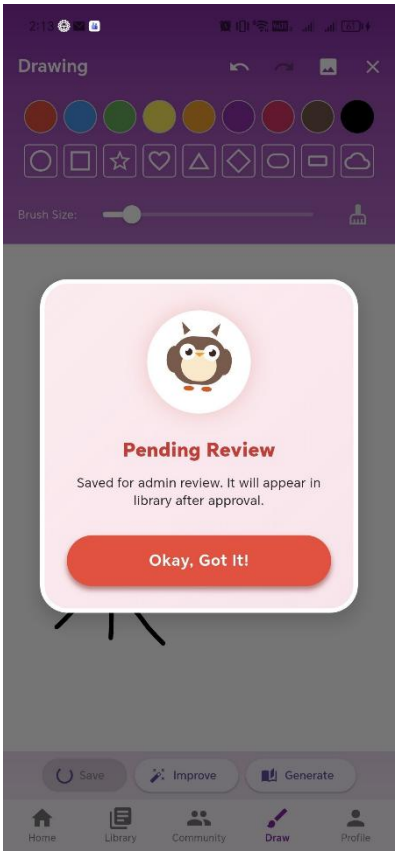


Figure 5.7.6 Pending Review Reminder

5.8 Genre Selection Screen

The genre selection screen as shown in Figure 5.8.1 appears as an intermediate step in the story generation workflow, activated immediately after users complete their drawing and choose to generate a story from their artwork as shown in Figure 5.8.2. This screen presents users with nine distinct genre options; each carefully designed to appeal to the target age group and provide clear visual and textual representations that help children understand the type of story they can create. The available genres include fairy tale, focusing on magic and princesses; adventure, which features exciting journeys and exploration; fantasy, which discover magic and creatures; science fiction, focusing on space and future; comedy, focusing on funny and happy; friendship, focusing on relationships and social connections; any style, which focusing on surprise me elements; mystery, involving puzzles and discovery; and nature, emphasizing environmental themes and natural settings. Each genre is displayed as an interactive card featuring an icon or image that visually represents the genre's theme, accompanied by a brief description that helps users understand what type of narrative they can expect. This visual and textual guidance is particularly important for younger users who may be less familiar with genre concepts. The interface allows users to select their preferred genre by tapping on the corresponding card, which provides visual feedback through highlighting or selection indicators as shown in Figure 5.8.3. Once a genre is selected, users can proceed to story generation by tapping the "Let's Create My Story" button, or they can return to the drawing screen using the cancel option if they wish to modify their drawing or choose a different action as shown in. After the user chooses their genre and clicks the create story button, the system initiates a dual-layered background safety moderation process. Before the generation begins, the application utilizes the Google Gemini Vision API to perform a real-time safety check on the user's drawing to ensure it contains no inappropriate or harmful imagery. Simultaneously, the system validates the chosen genre and context against children's safety guidelines. If the content is flagged as high-risk as shown in Figure 5.8.4, the generation process is stopped, and a child-friendly safety notice is displayed as shown in Figure 5.8.5; otherwise, the system proceeds to create the personalized story. If the pending drawing for generate story gets approved by admin, then the user can go to library screen to view the drawing and perform generate story into it. During the story generation process, a loading indicator is displayed to inform users that their personalized story is being created by the system Figure 5.8.6. After the story is successfully generated, it will bring the user directly to the story detail screen.

Figure 5.8.1 Genre Selection Screen

Figure 5.8.2 Generate Story Button

Figure 5.8.3 Genre Highlight

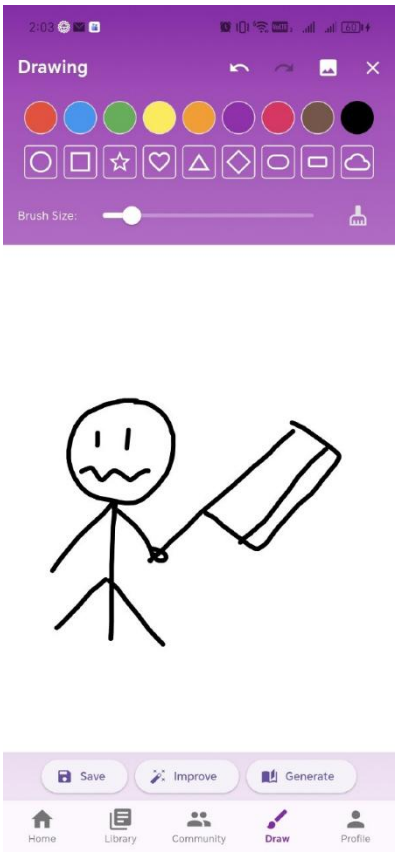


Figure 5.8.4 High Risk Content

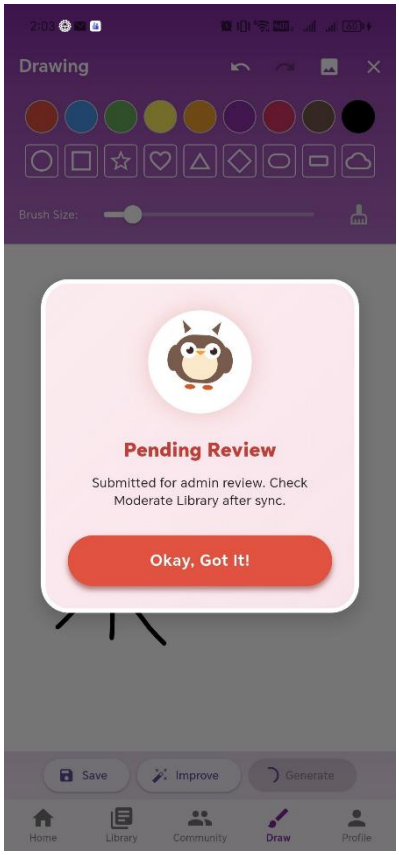


Figure 5.8.5 Safety Notice Reminder

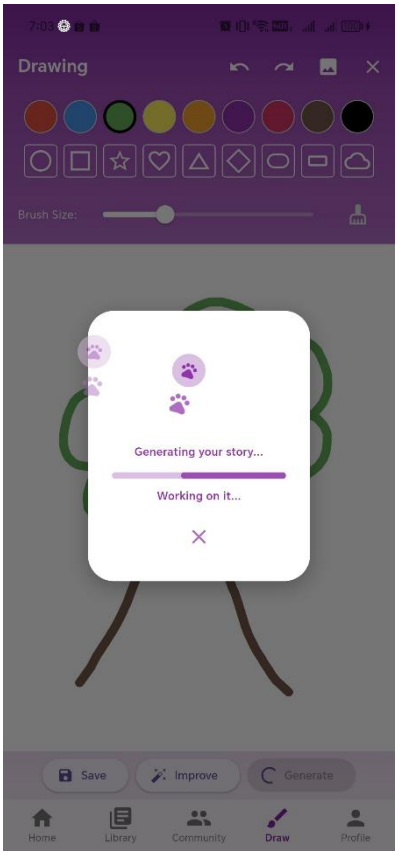


Figure 5.8.6 Loading Animation

5.9 Story Detail Screen

The story detail screen shown in Figure 5.9.1 provides a comprehensive overview of an individual story, presenting users with complete information about the story's content, metadata, and available actions in a single, organized interface. This screen serves as the central access point for story-related activities, displaying the story's cover image prominently at the top, followed by the title and a descriptive summary that gives users a clear understanding of the story's content and themes. The screen organizes story metadata in an easy-to-read format, including information such as the author, genre categories, recommended age range, estimated reading time, and creation date, providing users with all the context they need to make informed decisions about their reading choices. The screen as shown in Figure 5.9.2 detects reading progress to determine whether to display a "Continue Reading" or "Start from Beginning" button, enabling users to resume their reading journey from where they left off. For stories that were generated from user drawings as shown in Figure 5.9.3, the screen displays a preview of the original drawing, creating a connection between the creative process and the resulting

narrative. This screen presents a comprehensive set of action buttons that allow users to read the story, edit its content, share story to community, add or remove it from favorites, export the story to share it outside the application as shown in Figure 5.9.4, or delete it from their library as shown in Figure 5.9.5. For stories that were generated from user drawings, the screen displays a preview of the drawing in the cover image area, and if the drawing has been enhanced, the screen provides a version selector that allows users to toggle between the original drawing and the enhanced version as shown in Figure 5.9.6 and Figure 5.9.7. This version selector appears as two side-by-side buttons labeled “Original” and “Enhanced” with corresponding icons, enabling users to compare both versions and choose their preferred visualization. The cover image dynamically updates to reflect the selected version, and users can save either the original or enhanced version to their device gallery using the “Save to Gallery” button, which adapts its label to indicate which version will be saved based on the current selection. They can also generate the story from their drawing if they only improve the drawing in the previous step. They can also share their drawings for original and enhanced versions into community. The story detail screen for original drawing has edit drawing and edit story buttons, while for enhanced drawing screen has only edit story button. Since the user performs edit drawing action to edit the drawing that has enhanced version, the application will show an edit drawing reminder to user as shown in Figure 5.9.8. If the user clicks “Yes, Edit It.”, then the application will bring the user to the edit drawing screen to perform editing. After the user performs edit on the drawing and saves it, the enhanced drawing will disappear and only leave the original drawing as shown in Figure 5.8.9. For the user who generates a story from drawing screen, the story detail screen will only show the edit story button but not show the edit drawing button as shown in Figure 5.8.3.

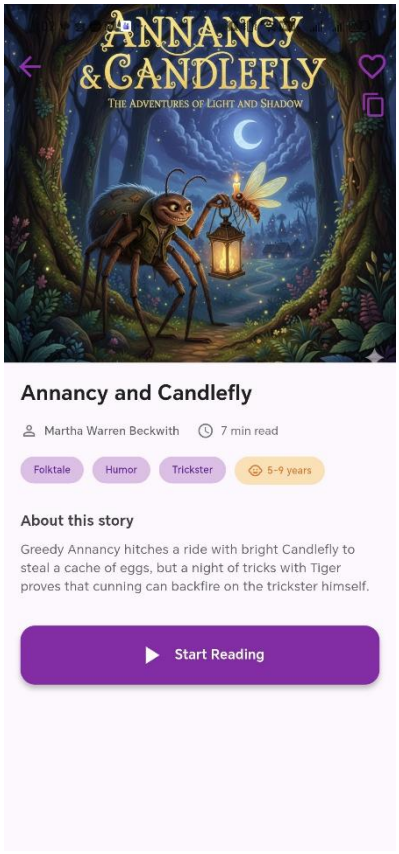


Figure 5.9.1 Story Detail Screen

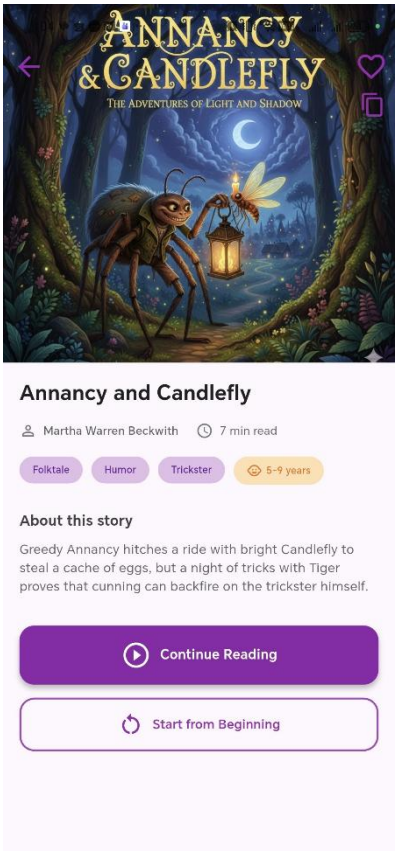


Figure 5.9.2 Continue Reading

Figure 5.9.3 Drawing-generated Story

Figure 5.9.4 Copy Story to Clipboard

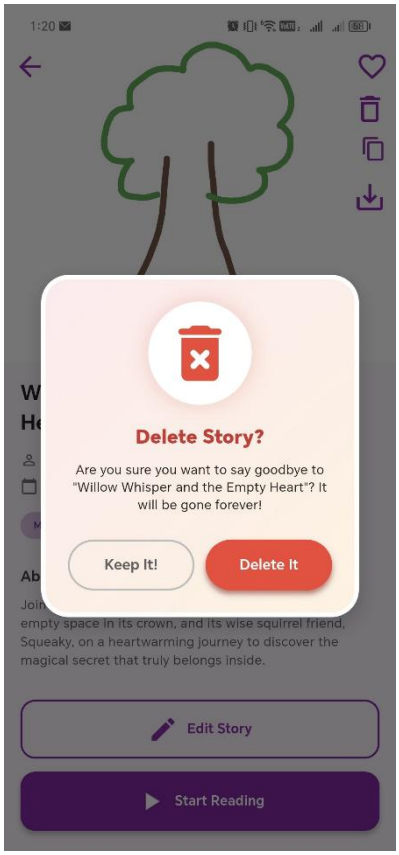


Figure 5.9.5 Delete Confirmation Dialog

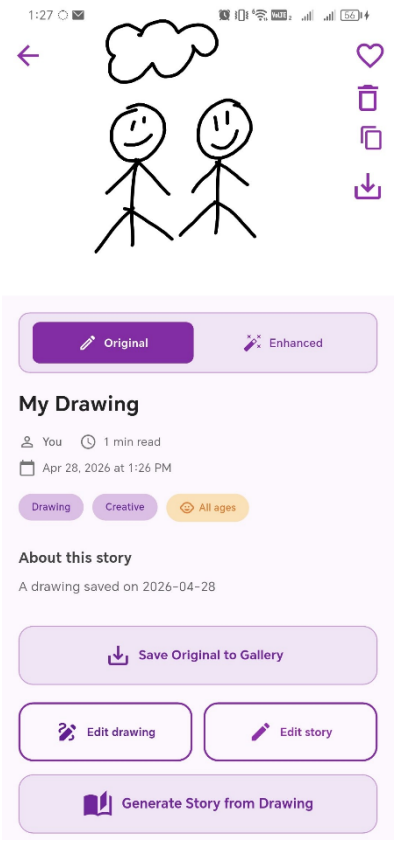


Figure 5.9.6 Original Drawing Selector

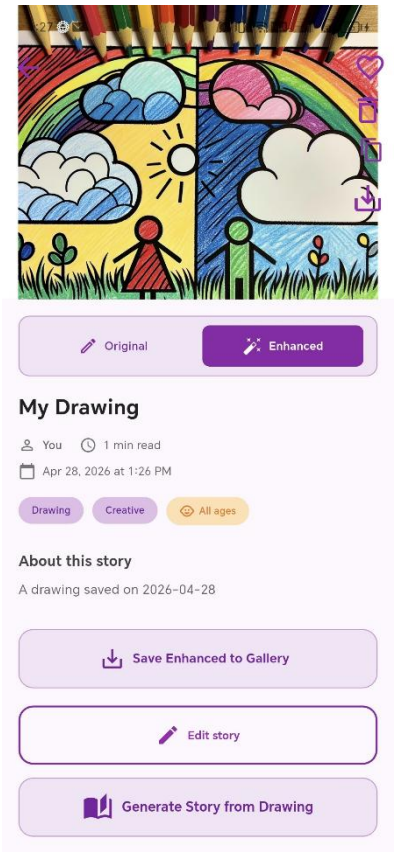


Figure 5.9.7 Enhanced Drawing Selector

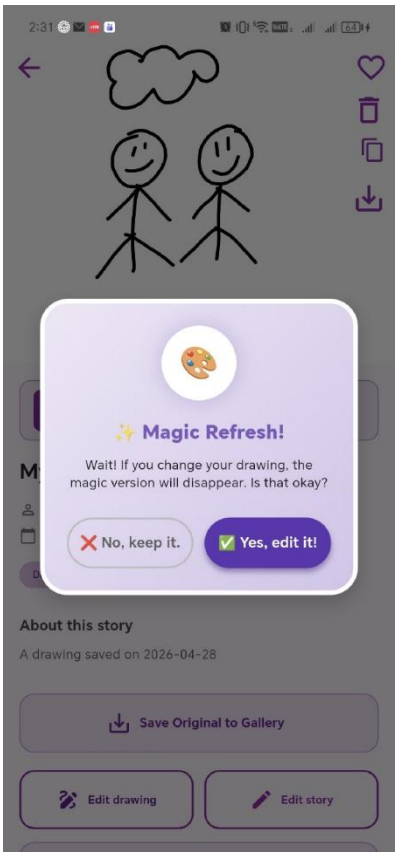


Figure 5.9.8 Edit Drawing Reminder

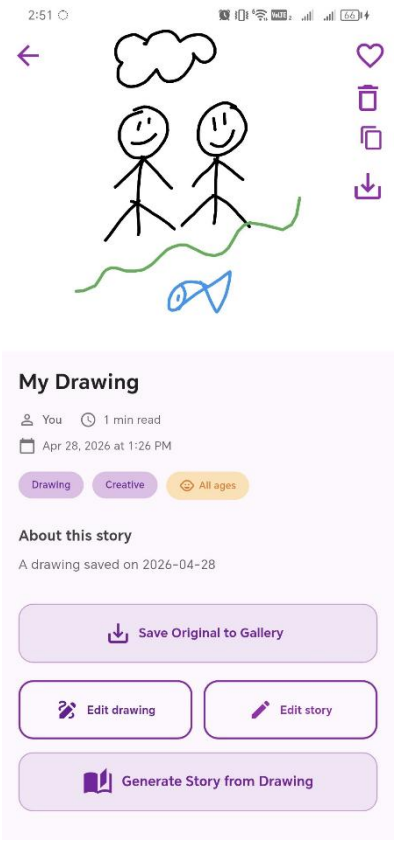


Figure 5.9.9 Edit Drawing Action

5.10 Story Reader Screen

The story reader screen as shown in Figure 5.10.1 provides an immersive, full-screen reading experience that has been specifically optimized for children, featuring a clean and distraction-free interface that allows young readers to focus entirely on the story content. The screen implements a page-by-page navigation system using Flutter's PageView widget, enabling users to progress through the story using intuitive swipe gestures or navigation arrows as shown in Figure 5.10.2, mimicking the experience of reading a physical book while leveraging the advantages of digital media. Each page displays the story text with carefully formatted typography that ensures readability and age-appropriate font sizing, accompanied by illustrations or images that enhance the narrative and provide visual context for the text. One of the screen's most significant features is its comprehensive Text-to-Speech (TTS) functionality, which supports multiple voice providers as shown in Figure 5.10.3. The TTS system includes sophisticated synchronization features that highlight words as they are spoken, helping children follow along with the narration and improving their reading comprehension. The system automatically scrolls to keep the currently spoken text visible on screen, ensuring that children can always see the words being read aloud. Users can control the narration through an intuitive control panel that provides play and pause functionality, speed adjustment options ranging from 0.5x to 1.25x speed as shown in Figure 5.10.4. The screen automatically tracks reading progress, saving the user's current page position when they exit, and displaying a page counter and progress indicator that helps users understand their position within the story. Additional features include adjustable text size for accessibility, night mode support as shown in Figure 5.10.5 for comfortable reading in low-light conditions, and smooth page transition animations that enhance the overall reading experience. The user is allowed to jump the page by typing the page number or scrolling the slider to jump to the wanted page as shown in Figure 5.10.6. Other than that, the reader screen includes an integrated dictionary support system designed for early readers. By tapping on any word in the story text, users can actively look up a dictionary lookup that displays the word's definition and pronunciation as shown in Figure 5.10.7. This interactive feature encourages children to expand their vocabulary independently while reading.

Figure 5.10.1 Story Reader Screen

Figure 5.10.2 Story Reader Button

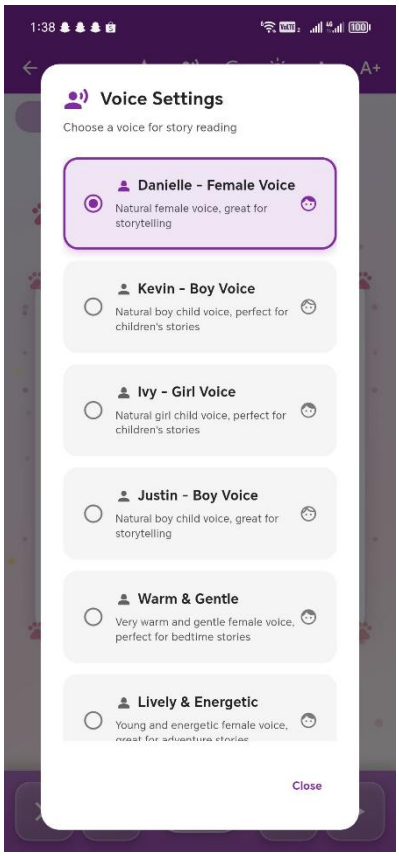


Figure 5.10.3 Voice Provider

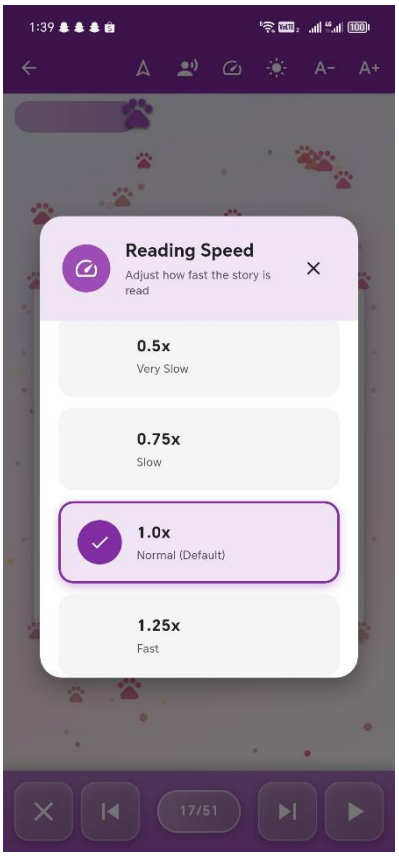


Figure 5.10.4 Speed Rate Sound

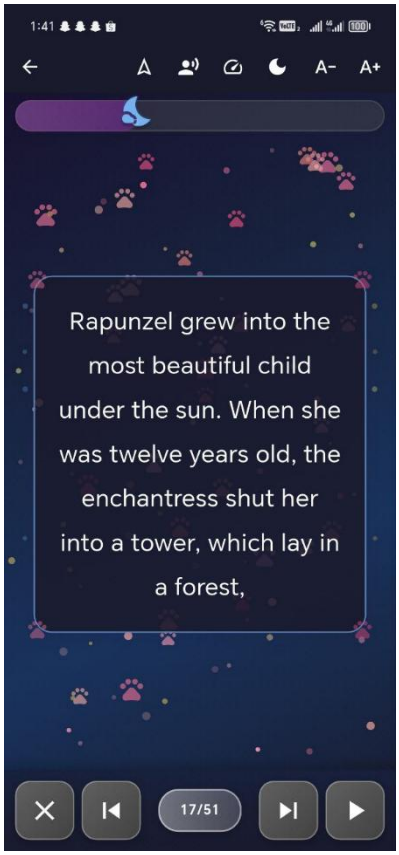


Figure 4.10.5 Night Mode Story

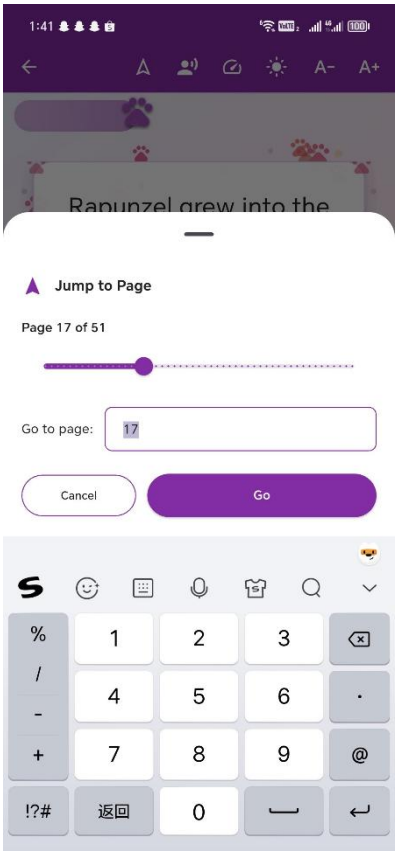


Figure 4.10.6 Jump Page Section

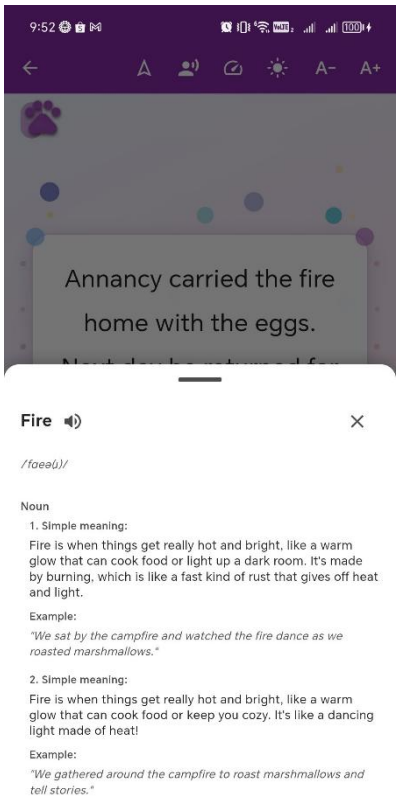


Figure 4.10.7 Dictionary Support

5.11 Story Edit Screen

The story edit screen as shown in Figure 5.11.1, Figure 5.11.2 and Figure 5.11.3 empowers users to always keep the contents safe and customize generated stories according to their preferences, providing a comprehensive editing interface that allows modifications to all aspects of a story including its title, description, and individual page content. This functionality recognizes that while generated stories serve as an excellent starting point, users may wish to personalize the narratives, correct details, or expand upon the creation to better align with their vision or address specific learning objectives. The screen presents a clean and organized layout with clearly labeled editable fields for the story title and description at the top, followed by a scrollable list of page editors that enable direct manipulation of each page's text content. The editing interface supports flexible content management through intuitive interactions that are designed to be accessible to users of various technical skill levels. Users can tap on any page within the list to activate editing mode for that specific page, making targeted modifications straightforward and efficient. The screen includes functionality to add new pages as shown in Figure 5.11.4, allowing users to extend their stories beyond the initial generation, and supports deletion of pages through swipe gestures, though the system prevents deletion if it would result in a story with no pages. Users can reorder pages through up and down interactions as shown in Figure 5.11.5, enabling narrative restructuring that supports creative storytelling workflows. The screen implements comprehensive validation that ensures data integrity, requiring that the title field cannot be empty and that at least one page contains content before saving. When users attempt to save invalid edits, clear error messages guide them toward resolution, while the cancel option allows users to discard changes and return to the previous screen without modifications. After the user click "Save Changes" button, the application will perform real-time safety check for story contents to ensure contains no inappropriate contents inside the story. If contains any inappropriate content inside the story, the story will be invisible in library and need to pass through admin review as shown in Figure 5.11.6. If admin approves the story, then the story will only be visible only in library screen.

Figure 5.11.1 Edit Story Button

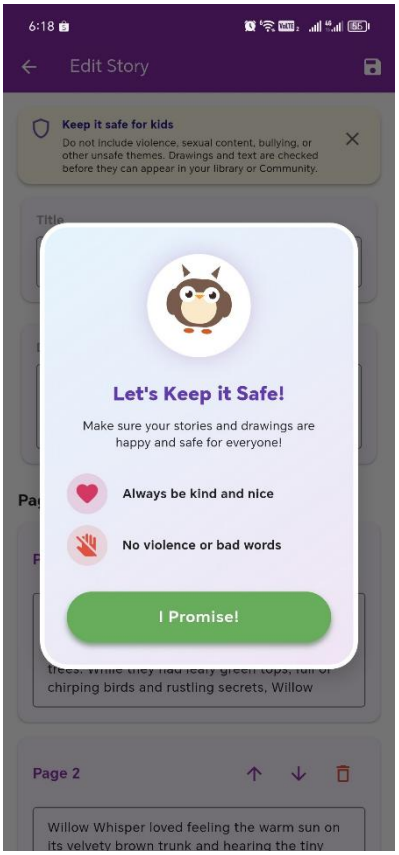


Figure 5.11.2 Keep Safe Reminder

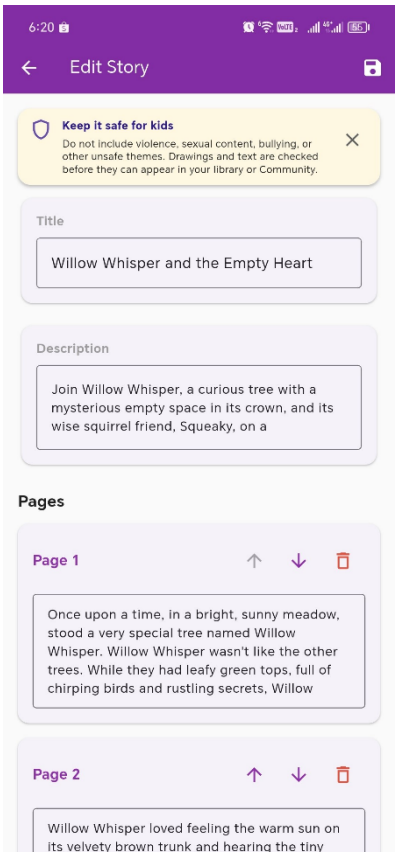


Figure 5.11.3 Edit Story Screen

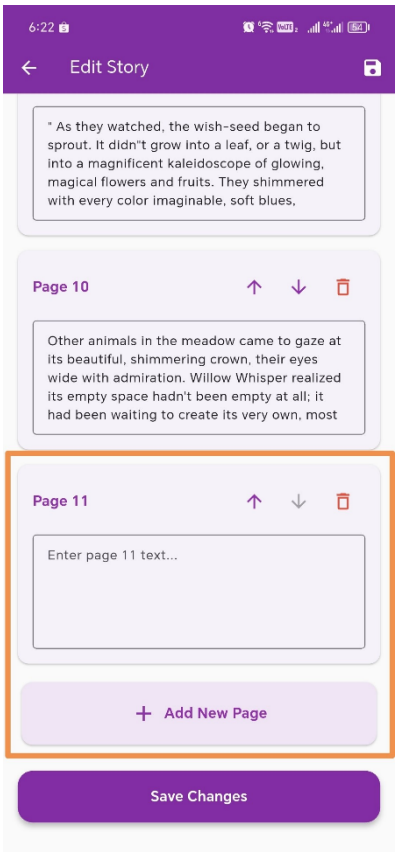


Figure 5.11.4 Add New Page Content

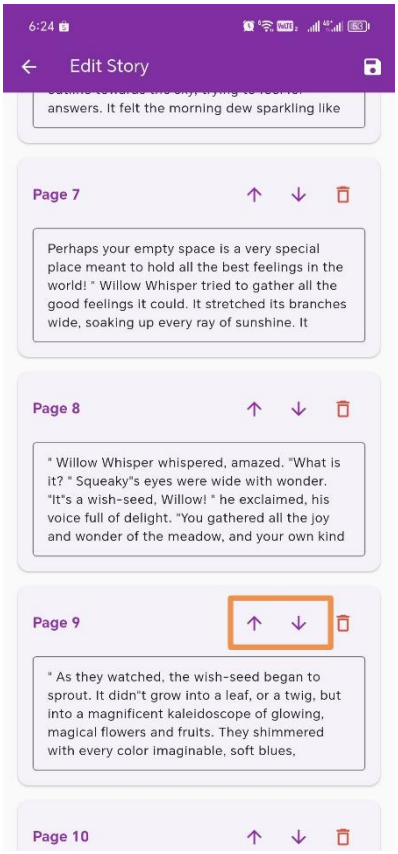


Figure 5.11.5 Up and Down Switch

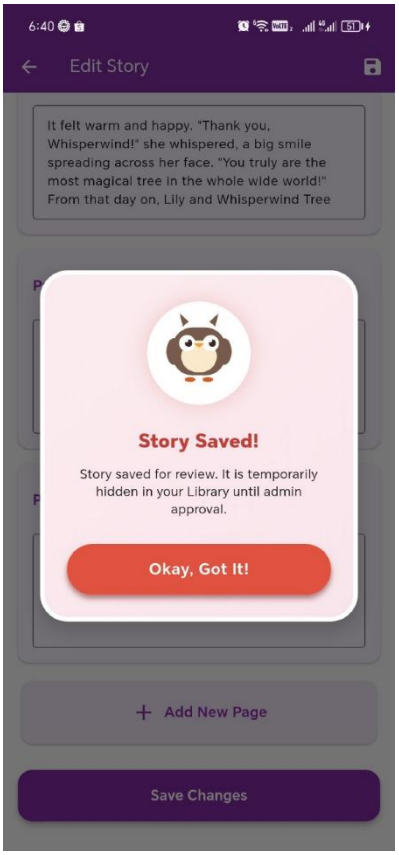


Figure 5.11.6 Unsafe Content Reminder

5.12 Edit Drawing Screen

The edit drawing screen as shown in Figure 5.12.1 provides users with a dedicated interface for revisiting and modifying their previously saved artwork. Unlike creating a new drawing from a blank canvas, the edit drawing screen is launched directly from the story detail screen by selecting the edit option on an existing drawing-based story. Upon launch, the system automatically restores the previous drawing session by loading the saved vector stroke data from local storage, ensuring that all original brush strokes, shapes, and colors are faithfully reconstructed on the canvas. If the original vector session data is unavailable, the system falls back to loading the saved cover image as a background reference, allowing the user to continue working from a visual representation of their previous artwork. The editing interface is identical in functionality to the standard drawing canvas, providing a full set of tools including freehand brushes, shape placement, a colour palette, eraser, and undo or redo controls. This design ensures a consistent and familiar experience for users returning to modify their work. Additionally, upon opening an existing drawing for editing, a child-friendly safety

reminder dialog is automatically displayed once per session as shown in Figure 5.12.2, reminding young users to create content that is kind and appropriate, reinforcing the application's commitment to a safe creative environment. Once the user has completed their modifications, they may save the updated artwork, generate a new story from the revised drawing, or enhance the drawing using the AI-powered enhancement service. All saving operations trigger a fresh safety moderation check through the library safety moderation service, which re-evaluates the updated drawing before saving it to the library. If the revised content passes the safety check, the story record is updated with the new artwork, and the user is notified of the successful save. If the content is flagged, a child-friendly notice is displayed, prompting the user to revise their drawing.

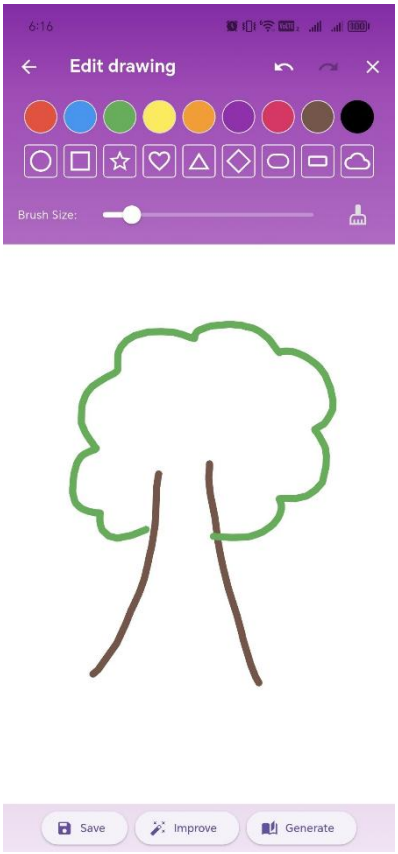


Figure 5.12.1 Edit Drawing Screen

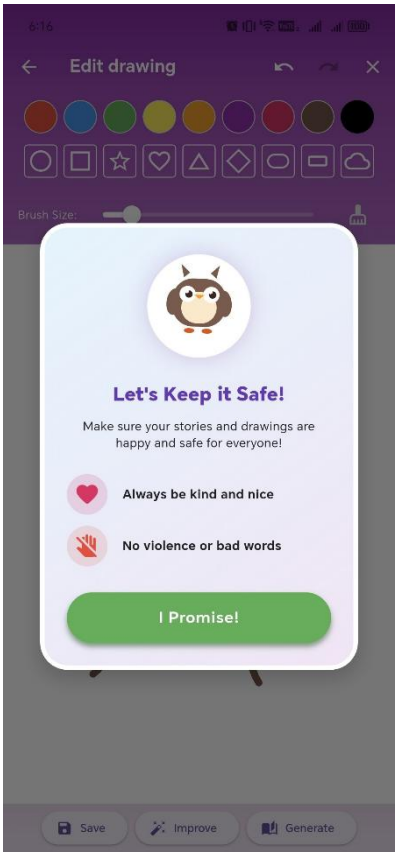


Figure 5.12.2 Safety Reminder Dialog

5.13 Profile Screen

The profile screen as shown in Figure 5.13.1 and Figure 5.13.2 serves as the user's personal dashboard within the StoryCraft application, providing a comprehensive overview of their activity, achievements, and account management options in a single, centralized location. This screen displays the user's profile picture and username prominently at the top, establishing personal identity within the application, followed by a visually engaging statistics section that presents key metrics about the user's engagement with the app. The statistics are organized in an easily digestible card-based layout that includes metrics such as the total number of stories read or created, the number of drawings saved, the count of favorited stories, cumulative reading time expressed in hours and minutes, story completion rate as a percentage, and a reading streak that tracks consecutive days of engagement. These statistics serve multiple purposes, such as providing users with a sense of accomplishment and progress, helping parents monitor their children's reading activities, and they gamify the reading experience through streak tracking and achievement visualization. The profile screen offers comprehensive profile customization options as shown in Figure 5.13.3 and Figure 5.13.4, allowing users to modify their username through an intuitive edit dialog that includes validation to ensure appropriate formatting and uniqueness, and to change their profile picture either by selecting from a collection of predefined avatars as shown in Figure 5.13.5 that are age-appropriate and diverse, or by importing a custom image from the device gallery. For authenticated users, the screen includes a logout option that requires confirmation to prevent accidental sign-outs, while unauthenticated users are presented with a login prompt that guides them toward account creation or authentication. Also, the user is allowed to change their password as shown in Figure 5.13.6. The profile screen has been expanded to include an "Account Status" section, which displays the user's current standing and any active content strikes. For administrative users, a "Quick Actions" panel provides direct access to the library and community moderation queues, enabling efficient management of the platform's safety directly from the profile dashboard. If the story or drawing is rejected by admin, the strike count will be increased by 1 for that user and the application will show the strike count security update notification in the "Account Status" panel and at the top of profile screen as shown in Figure 5.13.7 and Figure 5.13.8. It will only show in profile screen only if the user gets first strike count. If strike count becomes 3, the user account will be blocked permanently and need to create a new account to use the application.

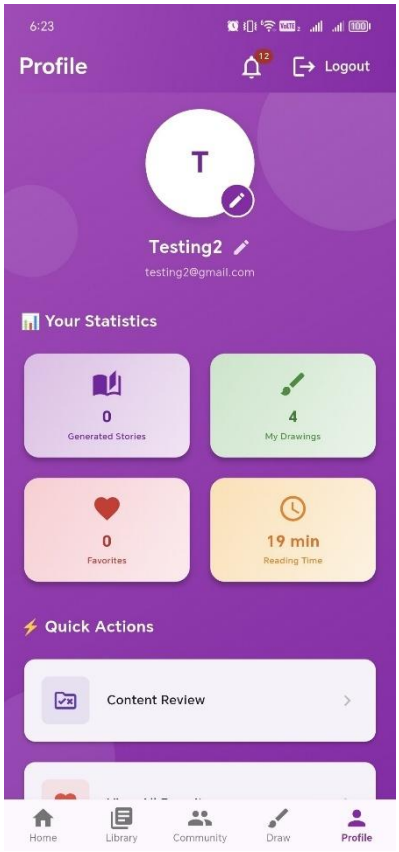


Figure 5.13.1 Profile Screen

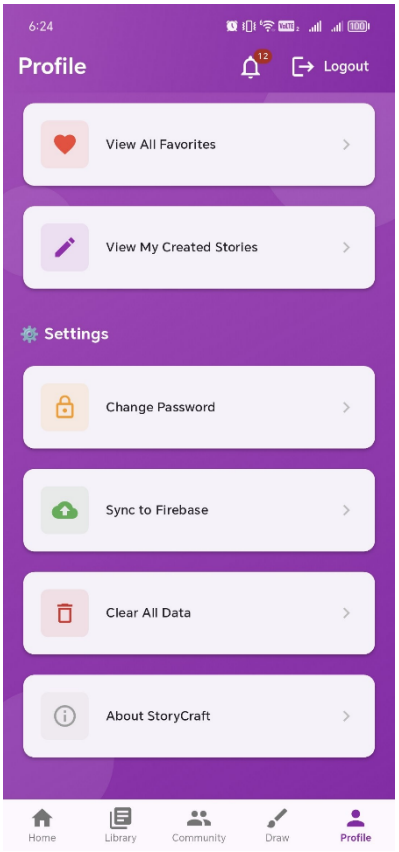


Figure 5.13.2 Profile Screen (Continue)

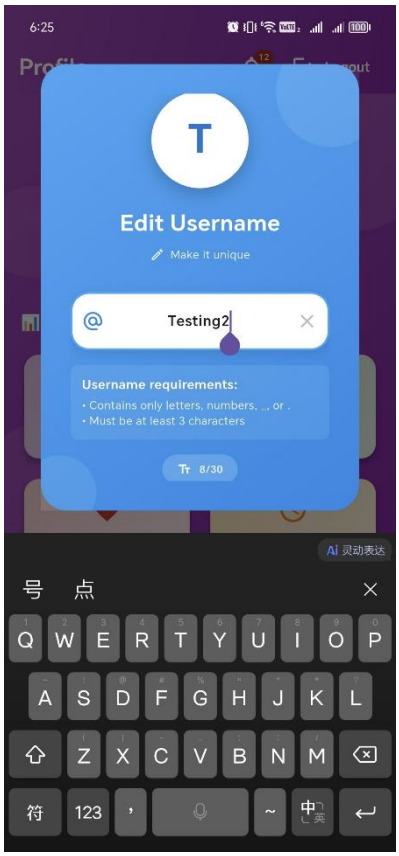


Figure 5.13.3 Edit Username

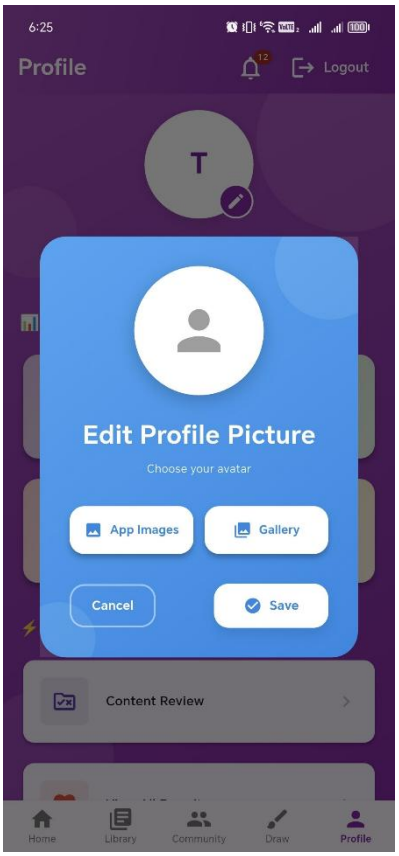


Figure 5.13.4 Edit Profile Picture

Figure 5.13.5 Predefined Avatar Picture

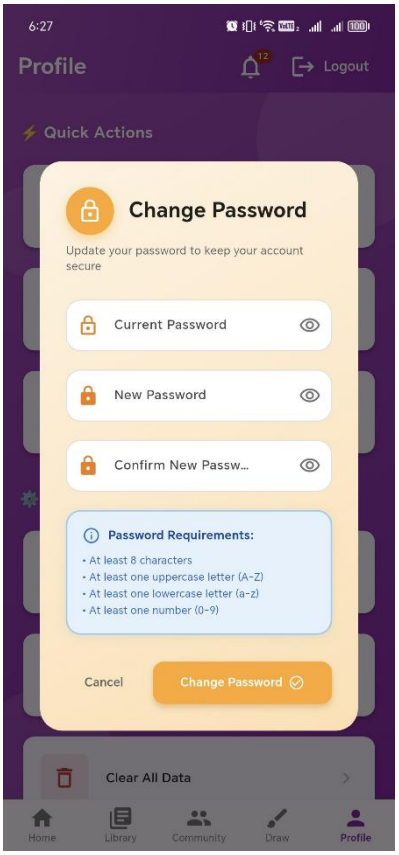


Figure 5.13.6 Change Password

Figure 5.13.7 Security Update Notice

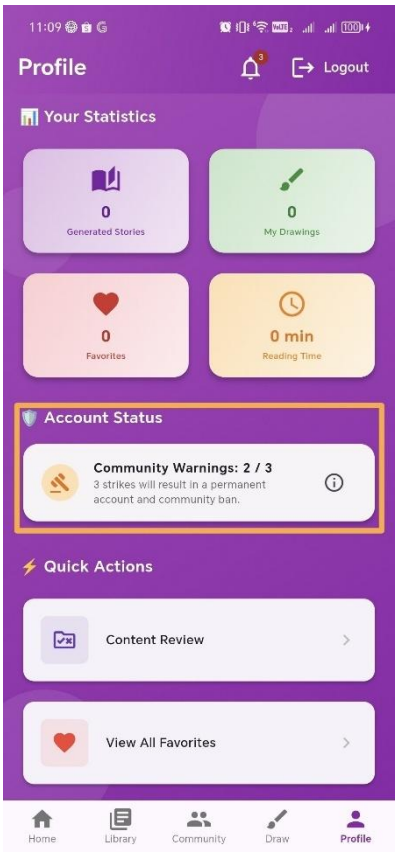


Figure 5.13.8 Community Warning Status

5.14 Community Screen

The community screen serves as the public center of the StoryCraft application, which provides dynamic social feed where the users can find and interact with stories shared by other creators as shown in Figure 5.14.1. Each community post is presented as a comprehensive card with the story's cover image, title, author name, and engagement metrics, such as likes and comments. These interfaces offer a clear and scrollable list of community posts. New posts and engagement updates appear instantly without requiring a manual refresh. Users can interact with these community posts by liking them, which provides immediate visual feedback through an animated heart icon, or by tapping the comment icon to participate in discussions. Other than that, the users can perform sharing features to share their posts with their friend within the app by clicking the share icon. This screen provides a filter feature for users to filter these community posts based on time order. The users also can click on the avatar or username of author to view their public posted posts. The users can also click "My Posts" as the top right of community screen to their preview their posted post in community. Through the community screen, the users can click the purple color message icon to access their friend chat screen. Besides, the users can choose to delete post action in their own posted post as shown in Figure 5.14.2 while performing report post action on the other user's post.

Figure 5.14.1 Community Screen

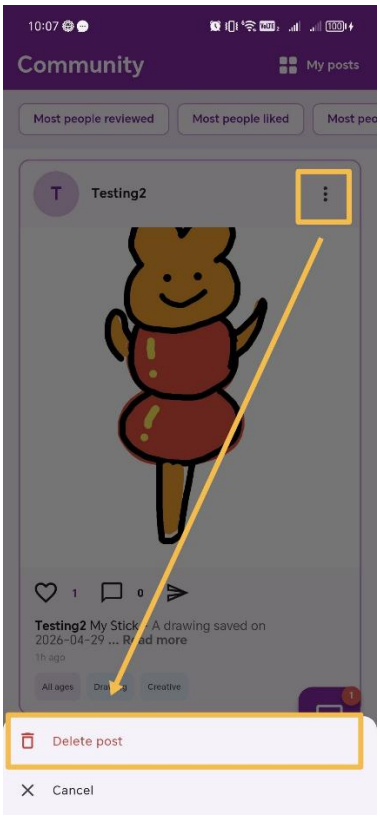
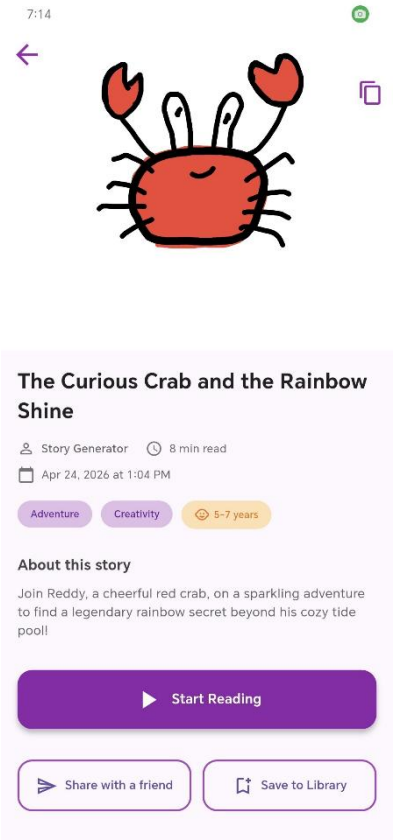


Figure 5.14.2 Delete Own Post Action

5.15 Community Detail and Comment Screen

The community detail screen provides an expanded view of a public post after the users click the highlighted “Read More” words in story description of the community posts in community screen, enabling users to read the entire story. Users can perform share post option by clicking the “Share with a Friend” button or “Save to Library” button to save the other’s stories into own library but cannot perform perform any edit feature on it. In the comment screen as shown in Figure 5.15.2 enables the users to type and submit their messages. Comments are displayed in a threaded format that supports replies, with each comment showing the user’s avatar and timestamp. To maintain a safe environment, the screen implements a real-time comment filter that checks content for bullying or profanity before posting the comment in that post. If the user posts any inappropriate content in comment, the application will show the reminder dialog for user to notify them about the comment is not allowed as shown in Figure 5.15.3. The users can perform delete on their posted comments if wrongly send it. Also, the admin account does have a delete all comment feature for all account within the StoryCraft application, ensuring that the discussion remains respectful and aligned with community standards. Users are also allowed to tag their friends in comments by using “@” symbol as shown in Figure 5.15.4.



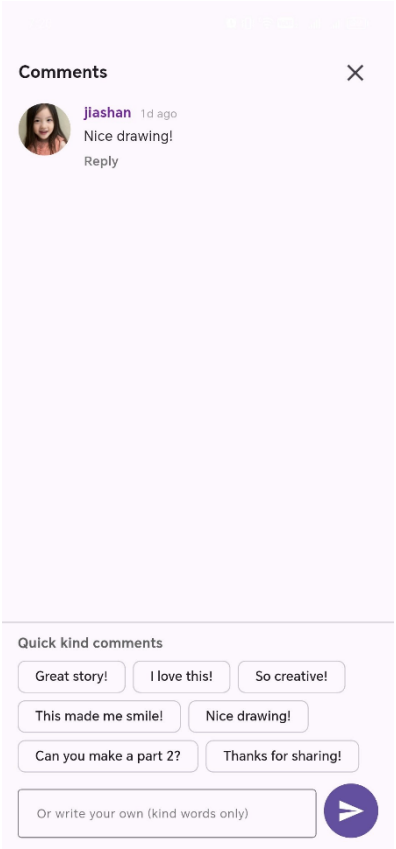
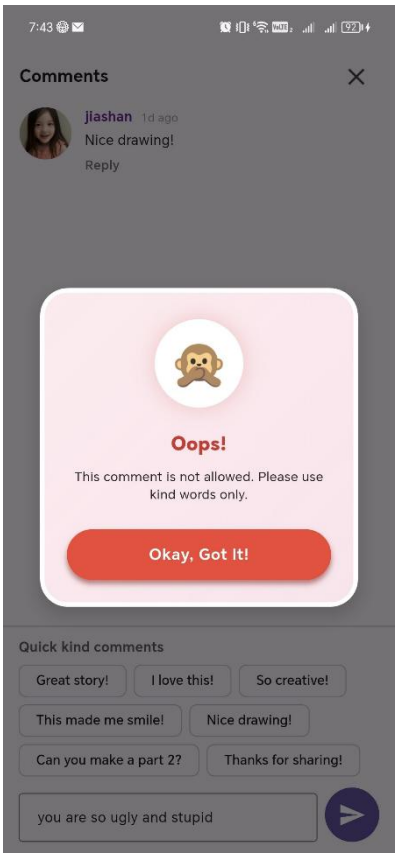


Figure 5.15.1 Community Detail Screen

Figure 5.15.2 Community Comment Screen



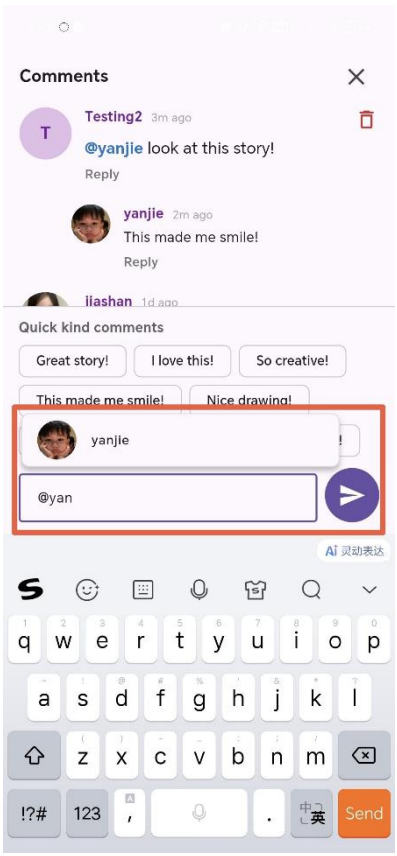


Figure 5.15.3 Disallow Content Posting

Figure 5.15.4 Tagging Username Feature

5.16 My Community Post Screen

My community post screen shows a personalized gallery for the user, showcasing all the user stories that have successfully passed the safety moderation process and are live in public feed as shown in Figure 5.16.1. this screen is designed to give users a sense of pride and ownership over their creative efforts, providing a curated view of their “Approved” social presence. The interface features a prominent profile header that displays the user’s avatar, username, and a total count of their active community posts. This screen also includes the sorting and filter features that let the user to view their posts using several criteria, such as “Most people liked”, “Most people commented”, as well as chronological sorting. This allows users to easily track which of their stories are performing best within the community. The screen uses the same visually engaging card layout as the main community feed, ensuring design consistency.

Figure 5.16.1 My Community Post Screen

5.17 Friend Conversations Screen

Friend conservation screen serves as the private messaging hub of the application, listing all active chat thread between user and their friend as shown in Figure 5.17.1. Each conservation entry displays the friend's profile picture, username, and a preview of the most recent message, along with an unread message badge that alerts users to new activity. Upon selecting a conservation, users are navigated to the chat screen as shown in Figure 5.17.2. Beyond standard text communication, the chat system allows users to share stories directly from the community post or community detail screen into the conversation. Shared stories appear as interactive cards within card bubbles, enabling friends to read and discuss specific stories in a private setting. For the shared stories, the users cannot perform any edit on it but can perform share with a friend and save to library. This screen supports real-time message delivery and reads status updates, providing a responsive and engaging communication experience.

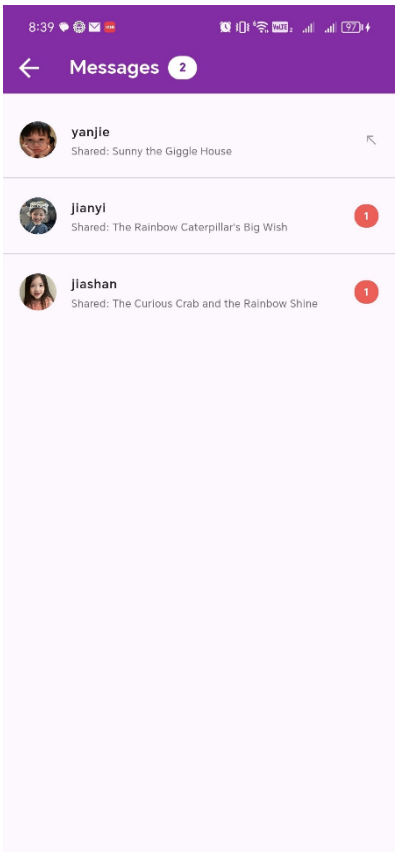


Figure 5.17.1 Friend Conservation Screen

Figure 5.17.2 Chat Screen

5.18 Library and Community Notification Screen

The library and community notification screen provides a centralized log of all social activities related to the user's account, accessible via the bell icon in the profile screen app bar as shown in Figure 5.18.1. The notifications are organized chronologically including alerts for new likes on post, comments, friend tags, and shares. Additionally, the screen screen serves as a critical communication channel for the moderation system, notifying users when their submitted stories have been approved or rejected by the review process. Each notification is interactive; tapping a like or comment notification navigates the user directly to the relevant post as shown in Figure 5.18.2, while moderation alerts provide feedback on content status. The interface includes a "Mark all read" option to enable user seen all notification as read status.

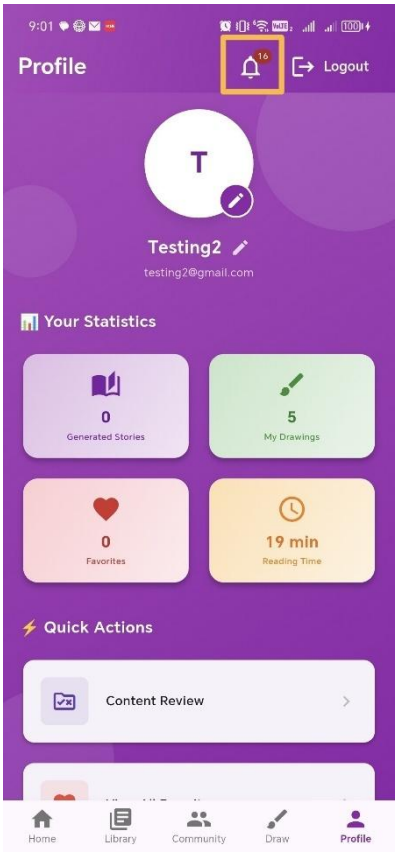


Figure 5.18.1 Bell Icon Notification

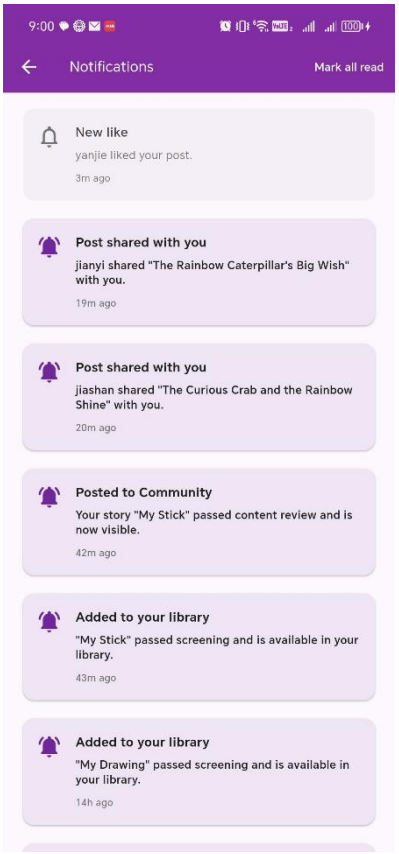


Figure 5.18.2 Notification Screen

5.19 Library Moderation Screen

The library moderation screen is an administrative tool designed to manage the safety of user-generated drawings and stories before they are finalized as shown in Figure 5.19.1. This screen can be found in admin account in quick actions bar. The interface presents a categorized queue of pending, approve, and rejected items, allowing admin to systematically review new submissions. Each item in the queue displays the user's drawing, preview story button and the AI-generated safety risk score. The screen provides "Approve" and "Reject" buttons, with the rejection action requiring the admin to provide a reason that is then sent to the user as shown in Figure 5.19.2. This screen ensures that all visual and textual content undergoes a final human verification step to maintain the highest safety standards for the target audience.

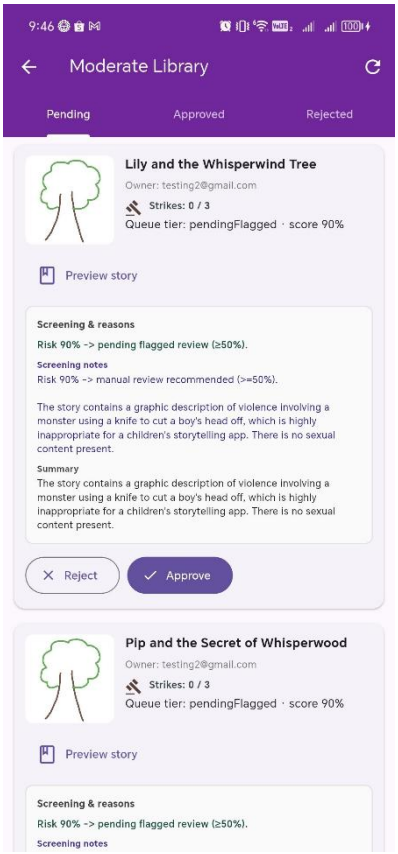


Figure 5.19.1 Library Moderation Screen

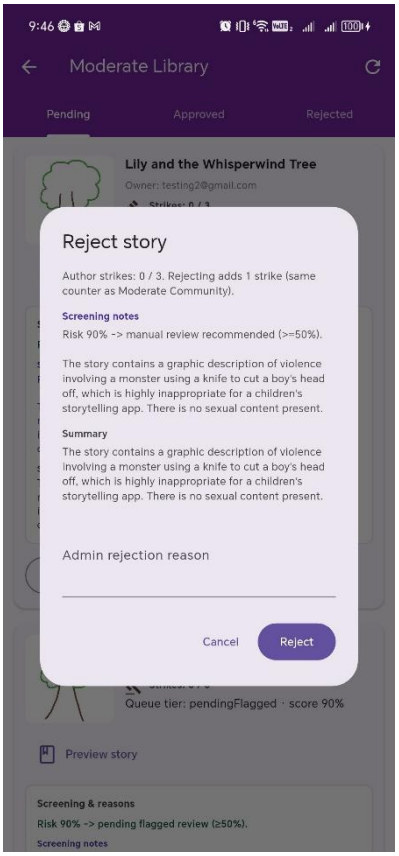


Figure 5.19.2 Reject Reason Action

5.20 Community Moderation Screen

The community moderation screen is an administrative interface dedicated to the management of user-reported content as shown in Figure 5.20.1. Unlike the library moderation queue which handles new submissions, this screen focuses on crowdsourced moderation, where the platform's community members act as the first line of defense by flagging inappropriate posts. When a post is reported, it appears in this queue along with the user's specific reason for reporting and original safety analysis. Admin can examine the reported content in detail and choose to either "Hide Post", which removes the story from the community feed and adds a strike to the author's account, or "Dismiss", which resolves the report if the content is found to be compliant with community standards. The screen also includes a strike counter for each author and a history tab to track previously resolved reports, ensuring that all disciplinary actions are documented and that the platform remains a safe and curated space for children. From policy compliance, many app ecosystems and school or child-focused deployments expect a user-report mechanism as part of safety governance.

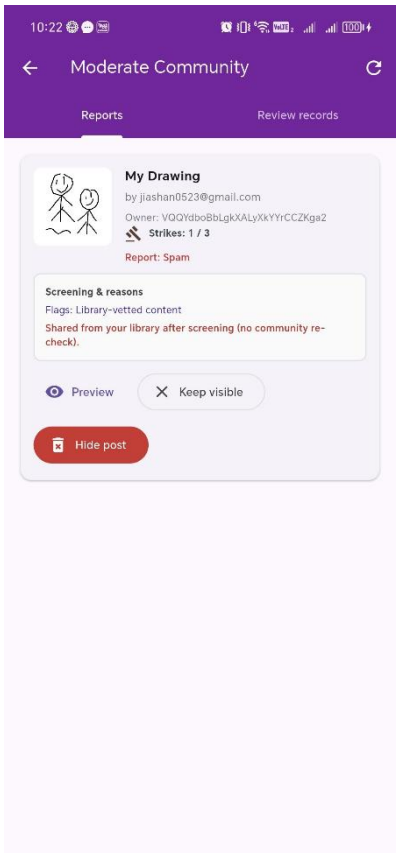


Figure 5.20.1 Community Moderation

5.21 Content Review Screen

The content review screen serves as a transparency portal for users, allowing them to track the status of their own generated drawings and stories as shown in Figure 5.21.1. The users can find this tab under the quick actions in profile screen. Designed with a clean, three-tabbed interface including pending, approved, and rejected, this screen ensures that users are always informed about the progress of their content through the safety moderation pipeline. Upon entering the screen, a child friendly safety reminder dialog is displayed as shown in Figure 5.21.2, reinforcing the app's community standards in a positive manner. Each tab provides specific feedback: the pending tab shows items currently awaiting review, the approved tab lists stories ready for public sharing, and the rejected tab provides clarity on content that failed to meet safety requirements. Each story card features a status badge with intuitive color-coding, such as orange for pending, green for approved, red for rejected. This screen is essential for maintaining user trust, as it provides a clear and organized view of how the system's inherited trusts and AI moderation models are being applied to their personal creations. For the stories which are pending approval by admin, the user cannot perform any action like editing on its story or drawing until admin performs approve or reject action on it as shown in Figure 5.21.3.

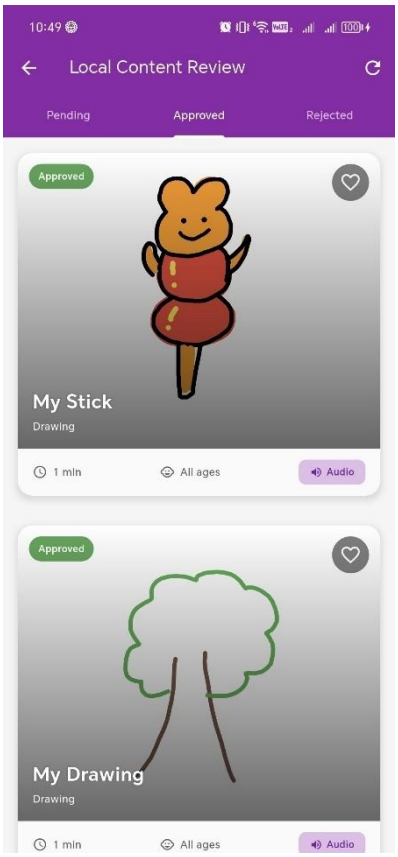


Figure 5.21.1 Content Review Screen

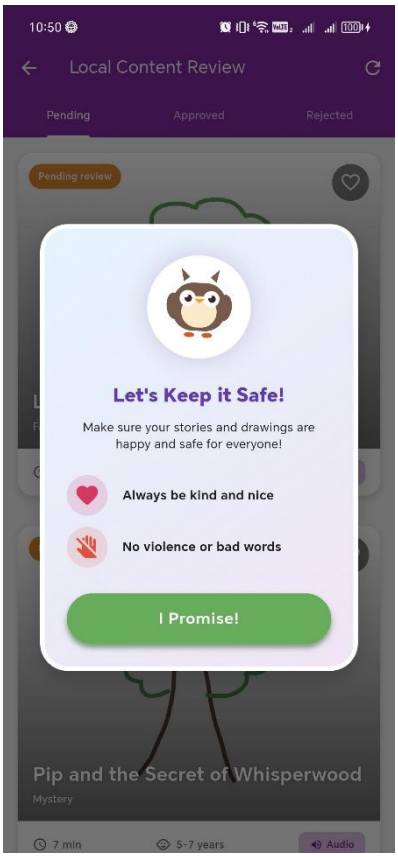


Figure 5.21.2 Safety Reminder Dialog

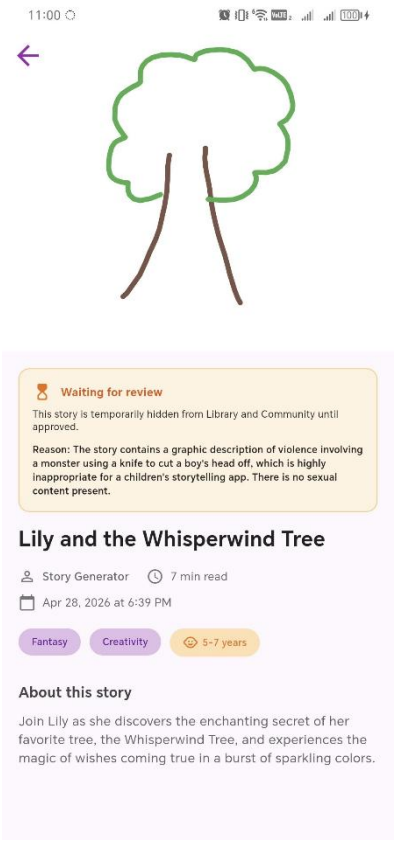


Figure 5.21.3 Waiting Review Story

5.22 Developer-led Functional Testing and Safety Validation

Due to the ethical considerations and data privacy regulations associated with the target audience of children aged 5 to 10, conventional user testing with minors was not feasible for this project. This phase used a black box testing methodology in which the application's features were tested completely form an end-user perspective. To ensure academic rigor, formal software testing techniques, including Use Case Testing, Decision Table Testing, and State Transition Testing were applied to validate both the standard application flows and the complex AI moderation logic.

5.22.1 Functional Integrity for Use Case Testing

Functional integrity testing was carried out to verify that all essential user flows functioned correctly from end to end. Each primary feature of the application was exercised through a series of predefined test scenarios using a use case testing. This included validating successful account registration, story generation from drawing inputs, and real-time community interactions. The stability of asynchronous operations, such as Firebase data reads and writes, was also validated by performing rapid sequential interactions to simulate real-world usage conditions. All primary user flows completed successfully without runtime errors as shown in Table 5.22.1.1, Table 5.22.1.2, Table 5.22.1.3, and Table 5.22.1.4.

Table 5.22.1.1 Functional Integrity Testing for Sign Up (UC-001)

Test ID	Covered Use Case	Test Action	Expected Result	Actual Result	Status
TC-01	UC-001 (Sign Up)	Enter valid email, unique username, and matching passwords	Account is created successfully and routed to Home Screen	Account created; user routed to Home Screen	Pass
TC-02		Enter an invalid email format like "user@com"	System blocks submission and show "Invalid Email" error	Registration blocked; "Invalid Email" message displayed	Pass
TC-03		Enter passwords that do not match	System blocks submission and	Registration blocked;	Pass

			show "Passwords much match" error	"Password must match" displayed	
TC-04		Enter a username that is already taken by another user	System checks database and shows "Username" is already taken	Registration blocked; duplicate username error displayed	Pass

Table 5.22.1.2 Functional Integrity Testing for Generate Story from Drawing (UC-005)

Test ID	Covered Use Case	Test Action	Expected Result	Actual Result	Status
TC-05	UC-005 (Generate Story from Drawing)	Draw an image and select a genre, then click "Generate"	AI analyzes drawing and generates story text in around 30 seconds	Story successfully generated and displayed in Reader Screen	Pass
TC-06		Submit an inappropriate drawing to the AI generator.	Vision API flags content and blocks generation	Generation blocked; child friendly safety warning displayed	Pass

Table 5.22.1.3 Functional Integrity Testing for Share Story to Community (UC-020)

Test ID	Covered Use Case	Test Action	Expected Result	Actual Result	Status
TC-07		Click "Share to Community" on a library story	Poss bypass AI check and goes live instantly	Post uploaded to cloud storage and appeared in feed	Pass
TC-08		Open a story that has a "Pending" moderation status	The "Share to Community" button is	Button is hidden; user is physically unable to share	Pass

	UC-020 (Share Story to Community)	in content review screen	completely hidden from the user interface	unapproved content	
TC-09		Click "Share to Community" on an approved story that has already been shared	System detected duplicate and shows "You already shared this story in community. Check it in My Community Posts"	Share blocked; duplicate warning message displayed	Pass

Table 5.22.1.4 Functional Integrity Testing for Add Comment (UC-022)

Test ID	Covered Use Case	Test Action	Expected Result	Actual Result	Status
TC-10	UC-022 (Add Comment)	Type a positive comment like "Great drawing!"	Comment passes AI gate and appears under the post	Comment published successfully via firebase stream	Pass
TC-11		Type a comment containing obfuscated profanity like f**k	Comment is blocked instantly; shows child friendly error	Comment rejected; "Please use kind words only" displayed	Pass
TC-12		Type a comment containing a web link like URL	Comment is blocked to prevent external tracking or spam	Comment rejected; "Links are not allowed in comments" displayed	Pass

5.22.2 AI Content Moderation Validation for Decision Table Testing

The most important component of the testing phase was the adversarial validation of the "Human-in-the-loop" AI moderation system powered by Google Gemini 1.5 Flash. To systematically test the logic gates that control content visibility, decision table testing was applied. The system was tested against various inputs, including obfuscated profanity, leetspeak, and inappropriate drawings to ensure the system correctly routed content based on the AI's calculated risk score.

Table 5.22.2.1 Decision Table for AI Content Moderation

Conditions	Test Case 1	Test Case 2	Test Case 3	Test Case 4
AI Risk Score	< 20%	20-49%	>=50%	Any
Is User Banned?	No	No	No	YES
Actions				
Auto approved	Yes	No	No	No
Send to Pending Queue (Yellow Flag)	No	Yes	No	No
Send to Pending Queue (Red Flag)	No	No	Yes	No
Block Submission Entirely	No	No	No	Yes
Test Status	Pass	Pass	Pass	Pass

5.22.3 Strike System and Ban Enforcement for State Transition Testing

To maintain long-term community safety, the application enforces a strict disciplinary system. To verify that repeat offenders are correctly penalized, the system was evaluated using State Transition Testing. The testing confirmed that the application correctly tracks moderation strikes within the user's database document. Upon reaching the critical threshold of three strikes, the system automatically transitions the user to a "Banned" state. Furthermore, testing confirmed that the system successfully executes a cleanup protocol, automatically hiding all previously approved posts from the banned user to protect the community feed. This transition logic is detailed in Table 5.22.3.1.

Table 5.22.3.1 State Transition Testing for Strike System

Current State	Event	Next State	System Action	Status
0 Strikes (Normal)	Admin rejects user's content	1 Strikes	Display Warning Message on Profile Screen	Pass
1 Strikes	Admin rejects user's content	2 Strikes	Display Warning Message on Profile Screen	Pass
2 Strikes	Admin rejects user's content	Blocked / Banned (3 Strikes)	Automatically hide all old posts; Revoke posting access	Pass

5.23 Implementation Issues and Challenges

One major programming challenge involved managing the application state during long-running network requests. The AI story generation phase requires approximately 30 to 35 seconds to complete. During early development, if a user navigated away from the loading screen prematurely, the application would crash due to memory leaks. This was resolved by implementing robust mounted checks across the Model-View-Service (MVS) architecture to ensure UI updates only occurred if the screen was still active.

Another challenge involved transitioning to a community platform required uploading local device images like cover images and canvas drawings to Firebase Cloud Storage. Initially, uploading high-resolution images caused network timeouts and delayed the community sharing feature. This was mitigated by implementing an image compression pipeline before uploading, ensuring payload sizes remained under optimal limits to achieve the current 5 to 8 second sharing latency.

5.24 Summary

In summary, Chapter 5 presented the complete implementation of the StoryCraft application, demonstrating the successful integration of creative drawing modules with a secure, AI-powered social community. Due to the ethical constraints of testing software on young children, the system was validated through rigorous Developer-led Functional Testing. By applying formal methodologies, including use case, decision table, and state transition testing. The functional integrity of the core workflows and the reliability of the Gemini AI moderation logic were thoroughly proven. Ultimately, the testing phase confirmed that StoryCraft is a highly stable and secure platform, fulfilling the technical requirements necessary for the final system evaluation discussed in Chapter 6.

CHAPTER 6 SYSTEM EVALUATION AND DISCUSSION

6.1 System Testing and Performance Metrics

System testing and performance evaluation were conducted to measure the efficiency, reliability, and responsiveness of the StoryCraft application under realistic usage conditions. Given the application's heavy reliance on cloud-based services, specifically Google Cloud Firestore for real-time data synchronization and the Google Gemini API for generative artificial intelligence and moderation, evaluating network latency and processing time was critical.

The performance metrics were established by measuring the execution time of core asynchronous operations over a series of trials. The primary metric of interest was the AI generation latency, which measures the total time elapsed from the moment a user submits a drawing to the moment the complete story is shown on the screen. This process involves image compression, cloud or payload upload, Gemini Vision API analysis, prompt engineering for child-safe context, and final text generation. Performance tests indicated an average processing time of 30 to 35 seconds. While this introduces a brief waiting period, it is necessary and acceptable latency given the computational complexity of multimodal AI generation. To reduce user drop-off during this period, the system employs engaging, animated loading dialogs.

Additionally, the AI moderation latency was evaluated. For real-time chat and comment submissions, the Gemini 1.5 Flash moderation gate processes text inputs in approximately 5 to 10 seconds. This processing time guarantees a thorough contextual analysis of the text to detect obfuscated profanity or implicit bullying before the data is committed to the database. Furthermore, the inherit trust model implemented in the community sharing feature significantly optimized overall performance because stories are already vetted when saved to the library, the system skips the Gemini AI safety check entirely when publishing. The measured latency of 5 to 8 seconds during the community sharing process is attributed exclusively to the necessary network upload of the local image assets like cover images and canvas drawings to Firebase Cloud Storage, ensuring the content is accessible to all users on the platform.

6.2 Testing Setup and Result

To ensure the accuracy and validity of the performance metrics and functional integrity, a controlled testing environment was established. The application was tested on both physical Android devices and high-performance Android emulators to simulate various hardware capabilities.

6.2.1 Testing Setup

Table 6.2.1.1 Testing Setup

Criteria	Explanation
Hardware	Physical Android (Honor 50).
Backend Environment	Firebase Cloud Firestore (NoSQL database), Firebase Storage, and Firebase Authentication.
AI Services	Google Gemini 1.5 Flash via REST API endpoints.
Methodology	Black-box functional testing and Adversarial Safety Validation as detailed in Chapter 5.22.

6.2.2 Testing Results

The results of the system evaluation confirmed the robustness of the StoryCraft architecture. The Model-View-Service architectural pattern effectively handled state changes during asynchronous operations without runtime crashes.

1. Safety and security: The adversarial testing resulted in a 100% success rate in identifying and blocking high-risk content like inappropriate imagery. The firebase security rules successfully prevented unauthorized administrative access.
2. Data integrity: Real-time listeners applied to the community feed and friend chat features demonstrated perfect consistency with UI states updating instantly upon database changes without requiring manual refreshes.
3. Resource Management: Image payloads sent to the Gemini API were successfully compressed to remain under the 1MB thresholds, preventing network starvation and ensuring stable animation frame rates in 60 FPS during the loading sequences.

6.3 Project Challenges

Throughout StoryCraft's development, several significant technical and architectural challenges were faced and successfully handled, especially during the shift from a standalone application in FYP1 to a fully moderated social platform in FYP2.

The first challenge involved AI prompt engineering and safety constraints. For instance, configuring the Gemini 1.5 Flash API to consistently produce age-appropriate, engaging narratives while strictly adhering to safety guidelines required extensive prompt iteration. The challenge lay in preventing the AI from generating overly complex vocabulary or mature themes, which was resolved by implementing strict system instructions and context boundaries within the API request.

The second challenge was balancing automation with human oversight. For instance, designing a moderation system that was both scalable and secure was complex. Relying solely on keyword blocking was insufficient for a children's app, leading to the integration of a "Human-in-the-loop" architecture. The challenge was developing a tiered risk-scoring system, including auto-approve, pending attention, and flagged that allowed the AI to filter obvious violations while routing ambiguous content to an administrative dashboard for final human judgment.

The last challenge was the limitation of the user acceptance testing (UAT). Due to strict ethical guidelines and data privacy regulations regarding the target demographic for children aged 5 to 10, traditional user testing could not be conducted. To overcome this limitation, the project successfully substituted UAT with Developer-led Adversarial Testing, proving the system's safety logic through rigorous black-box boundary testing.

6.4 Objectives Evaluation

The success of the StoryCraft application is measure against the primary objectives established at the beginning of the project. The final system evaluation confirms that all initial objectives have been successfully achieved. Below table show the project objectives.

Table 6.4.1 Overview of Project Objective

Number	Objective
1	To implement a child-friendly drawing module with visual assistance tools to support creative expression and ease of use.
2	To develop a personalized and creative storytelling feature that generates editable stories based on user-provided drawings or uploaded images.
3	To build a safe and moderate community platform for children to share, like, and comment on stories.

6.4.1 Evaluation for Objective 1

The evaluation for objective 1 was achieved. The application successfully features a highly interactive and intuitive drawing canvas designed specifically for children aged 5 to 10. To support users who may struggle with freehand drawing, the module includes visual assistance tools such as drag-and-drop shape placement and a comprehensive color palette. The interface utilizes large, accessible buttons and vibrant visual feedback to meet children's usability expectations. This environment successfully encourages motor skill development and allows children to effortlessly transform their imagination into visual art.

6.4.2 Evaluation for Objective 2

The evaluation for objective 2 was achieved. By integrating the Google Gemini Vision API, the application successfully bridges the gap between static drawing and dynamic narrative. Whether a child utilizes the in-app drawing canvas or imports an existing image, the system analyzes the visual input alongside a user-selected genre, such as mystery, fairy tale or adventure to generate a unique, personalized story. Furthermore, the inclusion of the story edit screen fulfills the requirement for interactivity, allowing children to actively edit and expand upon the AI-generated text, thereby fostering both creativity and language skills rather than relying on pre-written content.

6.4.3 Evaluation for Objective 3

The evaluation for objective 3 was achieved. The transition to a social platform in FYP2 successfully addressed the lack of collaborative spaces in traditional storytelling apps. The system features a fully functional community feed where users can publish their approved stories, leave comments, and interact via a like system. Most importantly, the safety of this environment was strictly enforced through a multi-layered moderation architecture. Instead of relying solely on basic keyword detection, the platform utilizes Gemini 1.5 Flash for deep contextual analysis of all comments and drawings, paired with an administrative dashboard for "Human-in-the-loop" oversight. This ensures that all interactions remain age-appropriate, fostering a positive and secure collaborative experience.

6.5 Summary

The development of StoryCraft represents a successful synthesis of mobile application engineering and Generative AI, evolving from a standalone creative tool in FYP1 into a fully moderated social platform in FYP2. By prioritizing user safety without compromising creativity, the project successfully achieved all its primary objectives.

CHAPTER 7 CONCLUSION AND RECOMMENDATION

7.1 Project Review, Discussion, and Conclusion

The primary objective of this project was to develop StoryCraft, an interactive, multimodal application designed to empower children to explore their creativity through drawing and storytelling. Over the course of the development lifecycle, spanning from initial prototyping in FYP1 to the deployment of a fully moderated social platform in FYP2, the project successfully evolved into a secure, real-time ecosystem.

The integration of the Google Gemini Vision API successfully bridged the gap between static user artwork and dynamic, personalized narratives. By allowing children to generate and actively edit stories based on their own drawings, the application effectively fosters both artistic expression and language development. Furthermore, the transition to a cloud-based architecture utilizing Firebase Cloud Firestore enabled the implementation of a real-time Community Feed and interactive Friend Chat. Developer-led functional and adversarial testing confirmed that the application operates with high data integrity, minimal latency, and zero runtime crashes. Ultimately, the project stands as a complete, stable, and highly engaging software solution that successfully meets all of its initial objectives and technical requirements.

7.2 Novelties and Contribution

StoryCraft introduces several significant novelties and contributions to the domain of child-oriented mobile applications. Unlike traditional storytelling apps that rely on pre-written, static content, StoryCraft utilizes Generative AI to provide a uniquely personalized experience for every user. Beyond its core creative features, the platform introduces two major architectural innovations regarding child safety.

The first one was human-in-the-loop moderation architecture. Most applications for children rely on simple, static keyword-blocking algorithms that fail to detect implicit cyberbullying or cleverly obfuscated profanity. StoryCraft introduces an advanced moderation tier powered by Gemini 1.5 Flash. This AI acts as a contextual filter, evaluating the intent of comments and drawings. Rather than executing automated bans, the AI routes ambiguous content to an administrative dashboard, allowing a human moderator to make the final disciplinary decision. This ensures maximum safety with minimum false positives.

The second one was inherited trust model. To optimize system performance and reduce cloud API computation costs, the application implements an "Inherited Trust" logic for community sharing. Content that has already been analyzed and approved during the local library generation phase bypasses redundant safety checks when shared to the community. This architectural contribution significantly minimizes network latency, allowing for near-instant community engagement.

7.3 Future Work

While StoryCraft is currently a fully functional and secure application, several recommendations can be made for future enhancements to improve scalability and user experience.

The first one is on-device machine learning integration. Currently, the application relies heavily on cloud-based APIs (Gemini) for both story generation and content moderation. Future iterations should explore deploying lightweight, on-device models such as TensorFlow Lite to handle initial moderation checks and basic drawing classification. This would drastically reduce API latency, lower operational costs, and allow the app to function partially offline.

The second one is real-world user acceptance testing (UAT). Due to strict ethical considerations and data privacy regulations regarding minors, the current system was validated via developer-led black-box testing. A critical future step would involve acquiring the necessary ethical clearances to conduct formal qualitative research and UAT directly with children aged 5 to 10. Direct observational feedback would provide invaluable insights into improving the user interface and accessibility.

The last one is voice-to-text and audio accessibility. To further support early childhood development, future versions of StoryCraft could integrate voice-to-text functionality for the interactive chat and comment sections. This would make the social features more accessible to younger children who have not yet fully developed their typing or spelling skills, thereby creating a more inclusive community environment.

REFERENCES

[1] E. Craft, B. McConnon, and C. Matthews, “Child-initiated play and professional creativity: Enabling four-year-olds’ possibility thinking,” *Thinking Skills and Creativity*, vol. 7, no. 1, pp. 48–61, Apr. 2012.

[2] L. Vygotsky, *Mind in Society: The Development of Higher Psychological Processes*, Cambridge, MA: Harvard University Press, 1978.

[3] Readmio s.r.o., “Whimsical Stories and Magical Sounds – About Readmio,” *Readmio*, 2025.

[4] V. Costenaro, “Language acquisitional storytelling: Psycholinguistics in action in the Italian EFL classroom,” *Studi di Glottodidattica*, vol. 2, no. 1, pp. 61–74, 2008.

[5] N. Purba, “The role of psycholinguistics in language learning and teaching,” *TELL Journal*, vol. 6, no. 2, pp. 79–92, 2018.

[6] Y. Yu, “The role of psycholinguistics for language learning,” *Frontiers in Psychology*, vol. 13, 2022.

[7] Y. Zeng, “Vocabulary instruction for English learners: A systematic approach grounded in psycholinguistic theory,” *Education Sciences*, vol. 15, no. 3, p. 262, 2025.

[8] Readmio, “About Readmio,” *Readmio Official Website*. [Online]. Available: https://readmio.com.

[10] A. Choudhury, "Software Development Life Cycle (SDLC) - Agile Model," *GeeksforGeeks*, Jan. 12, 2024.

[11] HQSoftwareLab, "How to Develop Software: Basic Methodologies," Blog of HQSoftwareLab, Apr. 2025.

[12] Eduteco Learning Games for Kids, *Short Stories for Kids to Read*, Google Play, 2025.

[Online]. Available: https://play.google.com/store/apps/details?id=com.eduteco.first.readers.

[13] Read 2 Me, *Read Aloud Library of Books for Kids* (Read 2 Me App). [Online]. Available: https://read2me.app.

[14] J. Piaget, "Piaget's Theory," *Piaget and His School*, vol. 1, no. 1, pp. 11–23, 1976.

[15] Ofcom, *Children and Parents: Media Use and Attitudes Report 2023*. London, UK: Ofcom, 2023. [Online]. Available: https://www.ofcom.org.uk

[16] UNICEF, *Growing up in a connected world*. UNICEF Office of Research, 2020. [Online]. Available: https://www.unicef.org

POSTER


STORYTELLING APPLICATION
STORY CRAFT
INTRODUCTION
• Storytelling is vital for early childhood development and imagination.
• Many existing apps restrict children to passive, template-based experiences, limiting their creative expression through drawing.
PROBLEM STATEMENT
• Lack of personalized and creative story generation for storytelling
• No drawing assistance and visual enhancement
• Absence of collaborative and community features
OBJECTIVES
• To implement a child-friendly drawing module with visual assistance tools to support creative expression and ease of use
• To develop a personalized and creative storytelling feature that generates editable stories based on user-provided drawings or uploaded images
• To build a safe and moderate community platform for children to share, like, and comment on stories
PROPOSED METHOD
• Flutter as cross-platform frontend development.
• RAD methodology used.
• Visual studio code as IDE to develop the application.
• Firebase Firestore as real-time storage and sync.
FUNCTION
• Login, Image to story generation, Drawing, History Tracking, Text-to-Speech (TTS), Story Editing and searching, User profile management, Drawing enhancement, and Setting and preferences
MOTIVATION
Transform passive digital experiences into an active, collaborative platform where children can generate, customize, and share unique stories directly from their own pictures and imagination.
UTAR
BACHELOR OF COMPUTER SCIENCE(HONS)
UNIVERSITI TUNKU ABDUL RAHMAN
DONE BY: LOH SHIN YEE
SUPERVISED BY: DR TAN JOI SAN