School of Electronic Engineering
and Computer Science

Final Report

Programme of study:
BSc(Eng) Creative Computing

Project Title:
Collecti - a tool to organise your digital world

Supervisor:
Dr Paulo Rauber

Student Name:
Jessica Malek

Final Year
Undergraduate Project 2024/25

Date: 25/04/25

Acknowledgements

I would like to thank Dr Paulo Rauber for supporting this project and giving feedback and advice throughout the duration of this project.

Abstract

In an era of constant discovery across multiple social media platforms, users are facing increasing difficulty in managing and retrieving saved content, which is scattered across platform-specific ecosystems. ‘Collecti’ is a mobile application developed to address this fragmentation by providing a unified space to organise, categorise and revisit saved content from various sources. It supports direct sharing of content into user-defined collections, integrates multiple social media APIs, and implements fallback embedding strategies to handle unsupported content. Built using React Native, with Firebase and Cloudinary used for storage, ‘Collecti’ employs a content-based recommendation system using natural language processing to suggest relevant collections. The system was rigorously tested through unit, integration and unmoderated remote user testing, with results indicating high usability and perceived usefulness. While limitations exist in platform API access and recommendation personalisation, ‘Collecti’ successfully demonstrates a scalable and user-centric solution for cross-platform content aggregation and organisation. Future enhancements are also proposed to deepen contextual relevance and expand third-party integrations.

Contents

Figures

Tables

Chapter 1: **Introduction**

1.1 Background

In the digital age that appears in front of us today, it is evident that individuals increasingly rely on social media platforms, particularly for content discovery, inspiration and personal interests. The average person engages with 6.7 different social media platforms, rising to 7.4 for younger demographics (DataReportal, 2024). While each platform often includes isolated features such as “collections” or “favourites” for saving content, these remain solely within their respective media ecosystem. This presented fragmentation means users face challenges when attempting to consolidate saved content from across multiple platforms/sources.

Currently, there is no seamless tool for cross-platform content organisation. Users must rely on manual saving methods or third-party solutions, which are often either incomplete or overly complex. For example, if a user wants to search for a specific cooking recipe, knowingly having saved many across platforms, they are forced to toggle between apps, causing inefficiency and frustration. Existing solutions fall short in integrating video content and lack robust sorting features, with true multi-platform compatibility.

1.2 Problem Statement

The proliferation of social media platforms has created a fragmented ecosystem for saving, organising and exploring general content and inspirations. Users are met with unproductivity resulting from the time consuming nature of manual content aggregation, due to the absence of a unified tool. This project focuses on the creation of a universal organisation tool, in the form of a mobile application. While the use case example centres on recipe organisation across platforms, the application's design is intentionally flexible, allowing it to be adapted for various interests and content types, hence streamlining the process of content aggregation.

1.3 Aim

The aim of this project is to design and develop a responsive mobile application that addresses this challenge of managing and organising saved content, particularly recipes, across social media platforms in one unified central space. The application will seek to solve this problem through offering a consolidated platform where users can save, organise, and manage content collections seamlessly, regardless of the origin platform. It will also include advanced sorting options, and the ability to create and manage custom collections.

1.4 Objectives

- Conduct comprehensive research to identify the gaps and limitations in the existing content saving tools across different social media platforms, and identify opportunities for improvement
- Implement API integration to enable direct content saving/retrieval from multiple social media platforms
- Design and develop a database schema for persistent storage of user data and content
- Create advanced tagging/sorting functionalities to facilitate organisation of saved content
- Develop a user-friendly, intuitive and aesthetically pleasing interface to enhance user experience
- Conduct testing to ensure reliability of API integration, database functionality and usability

1.5 Research Questions

How can an integrated mobile application streamline the organisation of content from multiple social media platforms into a single accessible space using API integration?

What are the key technical and design considerations for enabling seamless cross-platform content management?

1.6 Report Structure

The report is structured as follows:

- **Chapter 1** outlines the background, problem statement, aims, objectives, and research questions that ground the motivation for the project.
- **Chapter 2** critically reviews the current landscape of content aggregation tools, social media APIs, categorisation models, and design principles, identifying gaps the project addresses.
- **Chapter 3** details the functional and non-functional requirements derived from literature, user needs, and technical constraints.
- **Chapter 4** describes the research and development methodology adopted.
- **Chapter 5** presents the system design, including user flows, wireframes, and UI considerations.
- **Chapter 6** documents the implementation, focusing on platform integration, metadata extraction, recommendation logic, and backend services.
- **Chapter 7** provides an overview of the architecture and codebase structure.
- **Chapter 8** covers testing methodology, user testing results, and design iterations.
- **Chapter 9** evaluates achievements, critically assesses system strengths/weaknesses, and benchmarks against existing tools.
- **Chapter 10** concludes the report and outlines potential directions for future development.

Chapter 2: **Literature Review**

This chapter discusses existing applications and research on content management for social media. Key studies in content aggregation are reviewed for each social media as well as issues surrounding user authentication and categorisation are explored.

2.1 Social Media Content Aggregation Apps

Much of these mainstream social media apps are optimised for content discovery rather than comprehensive categorisation and organisation features. (Hu et al., 2014)(Zarro & Hall, 2012). The platforms rely heavily on exploration through visual cues , but create a number of challenges for users who wish to systematically organise their saved content.

2.1.1 Instagram

Instagram is a visual social platform centered around sharing image and video content. It provides basic functionality for saving content found in the app through its Collections feature, which functions like folders to group saved content.

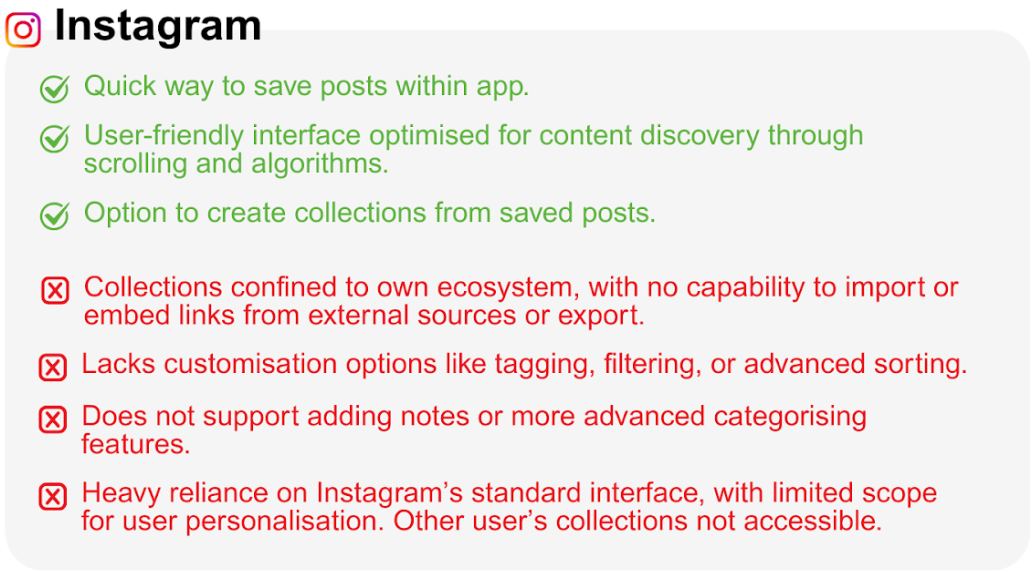


Figure 1.1: *Positives and Drawbacks of Instagram*

2.1.2 Pinterest

Pinterest is a visual discovery and bookmarking platform, whereby users can ‘pin’ images, videos etc. to their own boards:

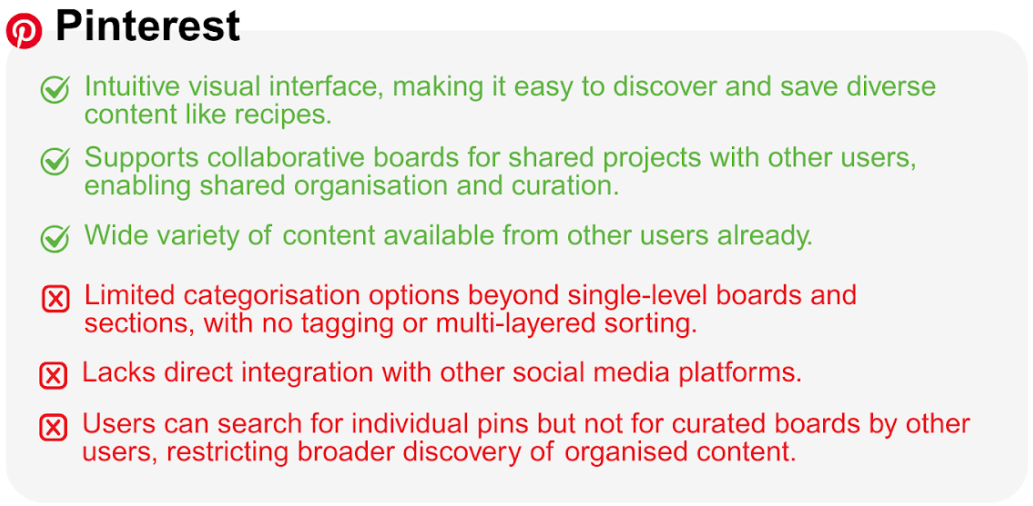


Figure 1.2: *Positives and Drawbacks of Pinterest*

2.1.3 Notion/Evernote

These are general purpose productivity tools designed for flexible, custom and manual organisation:

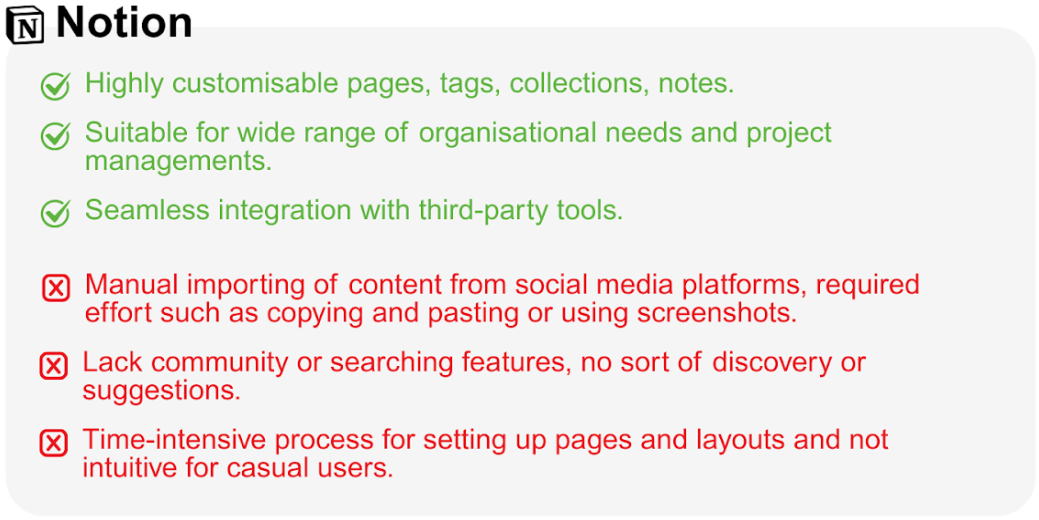


Figure 1.3: *Positives and Drawbacks of Notion*

2.1.4 Paprika

Paprika is a niche content management app specialised in managing and organising cooking recipes with a focus on manual user inputs:

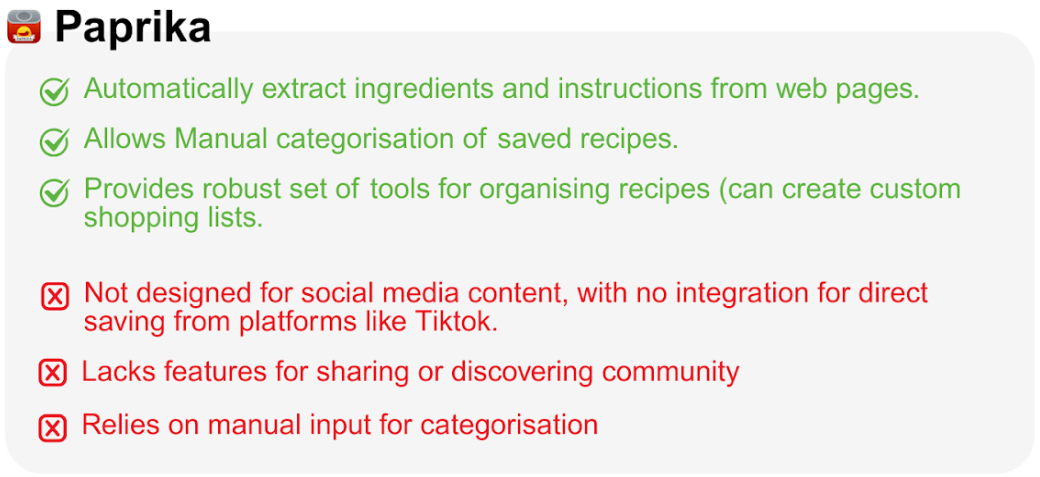


Figure 1.4: *Positives and Drawbacks of Paprika*

2.1.5 TikTok

TikTok is optimised for short-form video content browsing but not inherently an organisational tool:

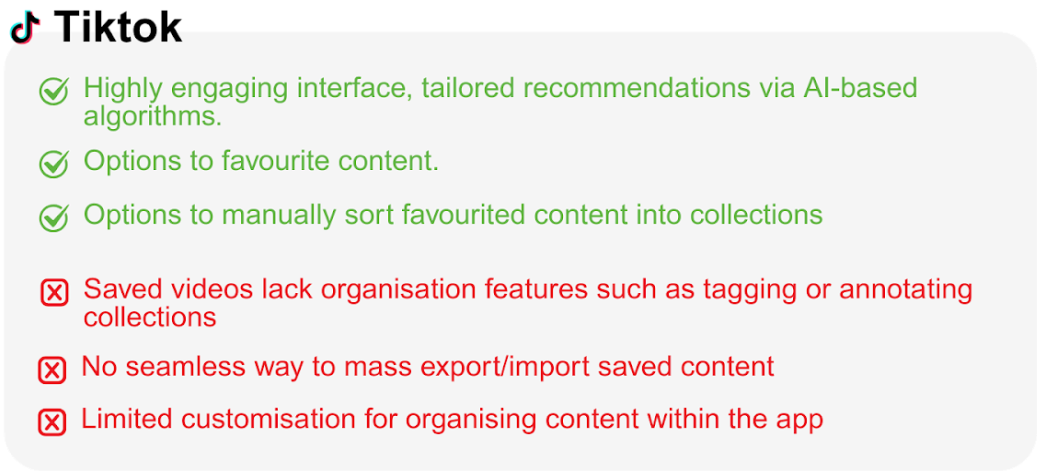


Figure 1.5: *Positives and Drawbacks of TikTok*

2.2 Social Media APIs for Content Integration

Integrating social media content into mobile applications requires navigating various platform-specific APIs, each with their own challenges such as access limitations, rate restrictions, deprecations and official and unofficial solutions. Requirements need to balance technical constraints with platform compliance, which can be challenging in small-scale projects, as major platforms enforce strict access controls over data to safeguard user privacy.

2.2.1 Instagram

Instagram offers multiple API solutions, each tailored to specific use cases but often with stringent requirements.

Basic Display API (Meta, 2024a): Enables retrieval of public profile data, photos and videos. While useful for basic integration, its scope is limited, and it is currently being deprecated in favour of the **Graph API**. This means it is no longer a suitable and reliable choice for data sourcing.

Graph API (Meta, 2024b): Primary API provided by Meta for accessing and managing Instagram data, being designed for business accounts to offer more advanced capabilities and access to media, comments, insights etc. It requires app review and business verification.

oEmbed API (Meta, 2024c): Allows embedding public posts by returning HTML markup through a single API call. However, this also mandates Advanced Access status, requiring business verification and a publicly available app.

Third-Party APIs (RapidAPI, 2024): Services like **Instagram Scraper API** offer unofficial workarounds by retrieving public data through web scraping techniques. These tools provide quick access to metadata, but are restricted by tight rate limits, with a free tier permitting only 30 requests per month. Although retrieving only publicly accessible data, scraping APIs also raise concern with compliance of the application's Terms of Service.

2.2.2 Pinterest

Pinterest API (Pinterest, 2024a) offers tools for accessing and managing Pinterest data programmatically, being divided into access tiers based on app approval and usage scopes:

Generated Access tokens: Restricted API access, allowing only limited, short-lived tokens (expiring every 24 hours), requiring manual renewal via the developer account. Such limitations are designed to safeguard user data to ensure operations are within platform guidelines.

Trial Access API: Provides more stable access and broader feature set, including access to user-created boards and pins. Requires app secret keys and app approval for access and extended use.

Full Access API: Grants access to advanced features such as analytics and broader scopes. However, this tier demands rigorous policy compliance and business verification.

2.2.3 TikTok

TikTok's API (TikTok, 2024a) caters to short-form video content, offering tools for developers to access, manage and analyse content. The API ecosystem focuses more on enabling user interaction within its platform, rather than on external content retrieval:

Core APIs: Include Login Kit, Share Kit, and Content Posting API, all of which are oriented towards sending data to TikTok - being highly effective for developers building applications that enhance user interaction within the TikTok ecosystem, but offering limited utility for external content aggregation/display.

TikTok Display API (TikTok, 2024b): Allows limited external display of a single user's profile and uploaded videos. The scope is narrow, making it unsuitable for aggregating content from multiple users or broader discovery.

Third-Party APIs (Teather D., 2024): Similar to Instagram, there are many third-party 'scraping' tools offering alternatives for retrieving TikTok content. These tools can fetch public video metadata but also face limitations such as rate caps and potential violations of platform policies.

Embedding Options:

Player Embedding (TikTok, 2024c): Leverages iframe to display TikTok posts with customisable parameters. Offers flexibility in appearance and how content is displayed within external applications, however there is no metadata access or interaction beyond basic playback.

Video Embedding (TikTok, 2024d)*:* Provides simple programmatic way to embed public videos through utilising oEmbed API, though it is limited to static display and requires app verification and policy compliance for advanced features.

2.2.4 Youtube

Youtube offers robust APIs for content integration, focusing on video access, data analytics and audience engagement.

Youtube Data API (Google, 2024a): The primary interface for interacting with YouTube content; supporting video retrieval, uploads and searches. OAuth2.0 authentication is required for certain endpoints, and API quotas apply. Developers must register for an API key through Google Cloud (paid service).

Youtube Embed API (Google, 2024b): Simplifies content embedding by providing HTML snippets. It avoids complex API calls but uses the default YouTube player with limited customisation.

2.3 Manual vs. Automated Categorisation in Content Management

2.3.1 Content Management

Content categorisation is a fundamental aspect of content management systems (CMS) which allow users to organise, search, retrieve content in an efficient and timely manner. A recurring challenge within the context of social media is the fragmentation of content collections across platforms. Users are often unable to aggregate topic-specific content from multiple sources, as highlighted by Pentina & Tarafdar (2014), which results in scattered collections that are difficult to manage and truly utilise. This fragmentation prevents users from maintaining a unified view of saved content, thereby impeding seamless content retrieval and management. This lack of a centralised repository complicates access as it is stored across disjointed sources. Bates' (1989) "Berry Picking" model of information retrieval illustrates the struggle users face to find and gather information when it is scattered across multiple platforms. Although originally applied to academic research and information seeking, the model is equally relevant to content collection. It characterises the process of content retrieval as non-linear, where users find themselves continuously revisiting sources and refining search criteria, which quickly becomes frustrating and inefficient when content is not easily retrievable. This reflects the fragmented experience of saving and locating content across various social media applications.

2.3.2 Manual vs. Automated Categorisation

Manual categorisation requires users to tag, label and sort content themselves by hand, which in turn provides high personalisation and context-specific organisation. This is demonstrated in recipe-specific tools like Paprika, where users can create custom tags to sort their saved recipes, or in Notion, which allows manual creation of bespoke structures such as pages, tables, and folders tailored to individual needs. This approach offers significant flexibility, as users are not constrained by predefined options and can adapt their organisational system to suit diverse content types. Users can apply categories that align exactly with their intended meaning, supporting precision and eradicating misconceptions. However, such categorising can be time-intensive. While it allows for personalisation, user-driven categorisation can result in inconsistent tagging practices, making it harder to retrieve content later and reducing overall system coherence.

In contrast, Automatic categorisation leverages machine learning (ML) and natural language processing (NLP) to directly assign categories or labels to content without direct user intervention. This method is common in applications such as Pinterest, where AI automatically classifies content based on metadata and patterns extrapolated from user interactions. Automation is efficient because consistency can be maintained, and large volumes of content can be categorised quickly, however AI may lack the nuanced contextual understanding that manual tagging provides, leading to inaccuracy. (Furnas et al., 1987).

More often than not, automated systems lack the personal touch that manual categorisation provides. This can be particularly important in applications which thrive on the categorisation and personalisation aspect. Automated methods may misinterpret content or fail to capture the subjective meaning that a human user would assign. A potential solution lies in hybrid systems, where automation provides initial suggestions to streamline the manual categorisation process, combining efficiency with user-driven accuracy.

2.3.3 Ontology in Categorisation

In the context of content categorisation, ontology can be seen as a structural representation of concepts (Wikipedia, 2024). Ontology can guide how content is classified and how users can search for it, through definition categories such as “recipe”, “desserts”, “snacks” etc. Relationships within the content allow for a richer and dynamic system of classification, meaning that both automated and manual categorisation can be presented as more intuitive and accurate (Heath & Bizer, 2011). Ontology implementation can assist users in classifying content according to specific criteria i.e content type, or user-defined tags, helping overcome the known issue of content fragmentation, by offering a unified framework.

2.4 User Authentication and Security in Mobile apps

User Authentication is a critical component of application development, particularly in mobile contexts where users frequently interact with sensitive personal data. A secure mechanism ensures that only authorised users can access specific resources, hence safeguarding both user privacy and application integrity. There are multiple approaches and services available for implementing authentication, outlined below.

2.4.1 Authentication Methods

Username and Password:
The most basic and conventional form of authentication is using a username and password, requiring users to provide a unique identifier (a username) and a secret password. While widely adopted and easy to implement across applications, it introduces security vulnerabilities due to password fatigue and re-use. Moreover, mandatory password creation can impact user retention as users are more likely to abandon apps if they are asked to create new credentials.

Two-Factor authentication (2FA):
2FA enhances security by combining something a user knows (e.g. a password), and something the user has (e.g. a phone or email address). While offering significant security upgrades and reducing the likelihood of unauthorised access, an additional login step like this can introduce friction, potentially lowering user retention rates.

OAuth2 (OAuth, 2024):
OAuth2.0 is an open standard for delegated authorisation, allowing users to grant third-party applications access without sharing credentials. This means users can for example have access to a single sign-on (SSO) implementation via a trusted app they have like Google *(Hardt, 2012).* This eliminates the need for remembering additional account credentials, which streamlines onboarding and adoption. However, it introduces a dependency on third-party services, meaning service outages or policy changes from providers could prevent users from logging in.

2.4.2 Authentication Services

Descope (Descope, 2024):
Descope provides an all-in-one solution for authentication and user management. It supports passwordless login, 2FA, and OAuth2 protocols, with SDKs that simplify integration and allow developers to customise UI components, without implementing backend infrastructure. Descope adheres to industry security standards (RH-ISAC, 2024) and reduces development overhead. However, it introduces a direct dependency, making the application vulnerable to service downtime or policy changes

Firebase (Google, 2024):
Firebase provides an authentication solution which integrates seamlessly with mobile applications. It supports multiple sign-in methods including email and password, phone number, and OAuth providers. Pre-built UI components, SDKs, and integration with other Firebase tools are also offered. While Firebase accelerates development and enhances scalability, it introduces the constraints of vendor lock-in. Reliance on this service also may raise concerns regarding data ownership and privacy, due to its integration with the broader Google ecosystem.

2.5 Design Principles for Organising Content (UX)

Effective content organisation is central to the usability, accessibility and overall experience of any digital application. This section explores the core principles of user experience (UX) and user interface (UI) design that inform how content should be structured and presented to prioritise intuitiveness, charity and user satisfaction.

2.5.1 UI Design

The way in which something is displayed and organised has a significant impact on how users navigate, understand and engage with content. Users rarely invest time in learning an interface, and instead, they expect intuitive navigation based on familiarity with prior experiences and interfaces (Krug, 2000). Krug also emphasises that users tend to visually scan interfaces during use, rather than directly reading them.This affirms that content needs to be well-structured and must employ visual hierarchies throughout, so that users can directly see and scan layouts. When content is organised, cognitive load is significantly reduced and users are more likely to enjoy and engage with an application. Furthermore, Krug warns against the “paradox of choice”, where excessive customisation options can overwhelm users and degrade decision-making efficiency. Therefore visual choices are good, but should also be limited for effective user guidance, and strong visual feedback mechanisms should be implemented to guide interaction and reduce the likelihood of user error.

2.5.2 Content Organisation in UX

Content organisation is not just about aesthetics and visual appeal, but rather it is about the creation of an experience where a user is able to quickly and seamlessly find relevant information - exactly what they’re looking for. By applying a visual and structural hierarchy, content can be prioritised effectively, supporting logical navigation. Categorising, whether through predefined types, metadata tagging, or ontology-based grouping, facilitates meaningful browsing experiences, allowing users to find what they search for with minimal effort. Shneiderman (2000) emphasises how universal usability is achieved when systems are designed to accommodate the diverse needs that users present, which include the idea of each user having varying levels of digital literacy. Employing consistency and predictability ensures that users, regardless of technical background, can build mental models of interactions, reducing the learning curve for an application that they have been introduced to (Krug, 2000). Norman (1988) also stresses the importance of aligning interfaces with users’ pre-existing mental models, leveraging familiar patterns so that they can feel intuitive in use. UI conventions like these help users anticipate system behaviour, reducing friction for new users.

2.5.3 User-Centric Design

User centric design is key to the creation of a seamless application, so it is essential to have understanding of both user needs and behaviour patterns. Accessibility features need to be put in place to accommodate varying needs, ensuring an app can be usable by everyone regardless of ability. Features to aid include readable fonts or sufficient colour

contrasts. Web content accessibility guidelines (WCAG) provide a standardised framework for ensuring compatibility and accessibility for users, achieving a user-centric design approach (W3C, 2018). This allows the app to become more than just functional - it can become a tool that users feel comfortable using and engaging with.

2.6 Research Summary

Existing solutions and social media platforms excel in the discovery of content, but lack robust tools for user-driven organisation and cross-platform content management. Content is often strictly within its own ecosystem, with limited affordances for filtering, categorising or long-term curation. This creates difficulty for the users that are seeking to actually manage and organise their digital content.

This project addresses this gap by proposing a unified solution that aggregates content from various social media platforms, allowing users to categorise, personalise and organise content in a centralised and intuitive manner. This integration will leverage platform APIs where possible and apply UX-driven design strategies such as clear navigation and meaningful visual feedback, to deliver a highly accessible and user-friendly experience. Ultimately, the solution seeks to overcome the current limitations and provide a scalable framework for digital content management.

Chapter 3: **Requirements analysis**

3.1 Functional requirements

The requirements for the system were identified through analysing existing tool limitations, evaluating competitors, and insights from stakeholder research. Functional requirements (Table 1) focus on the core functionalities necessary to ensure the system delivers tangible user value.

Functional Requirement	*Met*
1. **Content Aggregation:** System must be able to pull content from various social media platforms either via Embed or API. It should support image, video and text content	✓✓✓
2. **Content Categorisation:** Content should be able to be custom categorised into collections. Content should be sortable in these collections.	✓✓✓
3. **User Creation:** System must support account creation and login via standard email+password but also other OAuth integrations	✓
4. **User Profile Management:** User should be able to edit and manage their profiles	✓✓✓
5. **Search Functionality:** System must allow for users to search through collections by keywords, and search through saved content via keywords and/or tags.	✓✓✓
6. **Content Sharing and Saving:** Users must be able to share content directly into the app from other supported social media platform, allowing to save and manage posts from these combined platforms in their personal collections	✓✓✓
7. **Tagging and Annotation:** Users should be able to manually add/edit tags and notes to saved content for better organisation and retrieval	✓✓✓
8. **Bulk actions:** Users should be able to move and delete multiple posts at once within an collection for ease of management	✓✓✓
9. **Manual content upload:** Despite aim of sharing from other platforms, users should still have the option to manually upload content from their devices or via links	✓✓✓
10. **Link validation:** If a post is linked from an external source, the app should validate and check the link, as well as check for embed possibility.	✓✓✓

Table 1: *System's Functional Requirements and whether they have been met*

3.2 Non-Functional requirements

In addition to core functionalities, the system must meet several non-functional requirements (Table 2) to deliver a smooth and reliable user experience.

Non-Functional Requirement	*Met*
1. **Usability:** User interface must be intuitive and easy to navigate. The design should be responsive, providing a good experience across a variety of devices (smartphones, tablets etc.)	✓✓✓
2. **Performance:** The app must be able to load content and perform operations in a timely manner.	✓✓✓
3. **Scalability:** The app must be designed to handle large volumes of both content and users, with the database architecture being scalable.	✓✓✓
4. **Security:** The app should ensure secure handling of user data, particularly authentication details, using encryption in transit.	✓✓✓
5. **Reliability:** The app should have minimal crashes or errors, with solutions in place to prevent unexpected failures or data loss.	✓✓✓
6. **Accessibility:** The app should follow accessibility guidelines (WCAG etc.) to ensure usability for all.	✓✓✓
7. **Data Retention & Deletion Policy:** Users should be able to delete their accounts and all associated data permanently (GDPR principles)	✓✓✓
8. **Cross-Platform Syncing:** If a user is logged in on multiple devices, collections and saved content should remain synchronised and updated in real-time	✓✓✓
9. **Error Handling and Logging:** The system should have internal logging for errors, with appropriate fallback mechanisms	✓✓✓
10. **User Feedback and Confirmation:** After every major action, the user should be provided with immediate visual feedback.	✓✓✓

Table 2: *System's Non-Functional Requirements and whether they have been met*

Chapter 4: **Methodology**

4.1 Research

For validating the need of the proposed solution, a critical analysis of existing tools was conducted (Chapter 2).Platform-specific saving features (e.g. Instagram saved collections, TikTok favourites etc.) and general-purpose note-taking apps were evaluated. The conclusions drawn informed the project's definition and scope, leading to the creation of a project timeline (Appendix A), with a Gantt chart outlining major project milestones and progress. A risk assessment was also carried out. (Appendix B)

4.2 Development

The development of 'Collecti' followed an iterative, prototype-driven approach, informed by real-time testing, evolving technical constraints and user-centric feedback. Initial wireframes and feature prototypes, such as the share intent, were developed for grasping feasibility of the project. Features were implemented incrementally throughout, prioritising core flows and later working on supplementaries. This allowed continuous refinement and integration based on testing outcomes. GitHub was used for version control and task tracking. Despite not strictly following an agile framework, many agile principles were adopted through development such as incremental delivery, responsiveness to change and continuous interactions based on the evolving understanding of different technological constraints and insights.

4.3 Testing

Given this is an implementation-focused project, it was necessary to include various methods of testing that would also translate to real-world application usage. Unit testing was conducted through direct integration in the development cycle - isolated individual components were tested, ensuring expected functionality before integration. Manual end-to-end testing scenarios were conducted across different key flows such as onboarding. Informal usability testing was also conducted, receiving immediate feedback regarding bugs or UI/UX concerns from peers who had interacted with the prototypes from various development stages. A heuristic evaluation was informally conducted to identify common issues based on established usability principles. Finally, formal user testing was carried out in the final phase, providing authentic feedback, critical to the project's refinement and completion.

Chapter 5: **Design**

5.1 System Use Cases

This section outlines the essential features and behaviours of the proposed application. The use cases are derived from the functional requirements, detailing the core system functionality.

5.1.1 Whole System Use Case Diagram

The primary use cases of the system involve managing content from various social media platforms, organising into collections and providing seamless user authentication. (Figure 2).

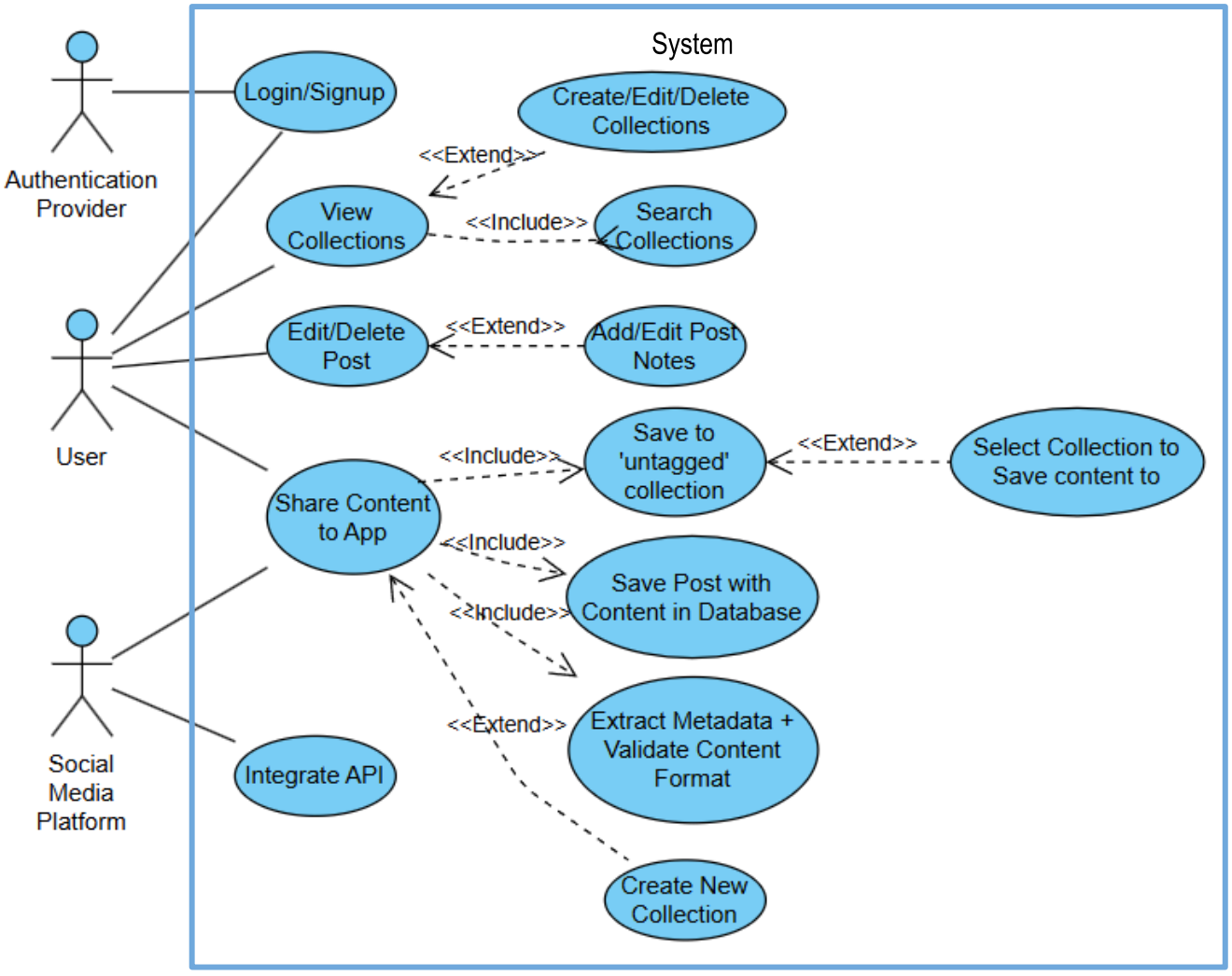


Figure 2: *Main Use Case Diagram of the System*

5.1.2 Detailed Use Cases:

Below is a detailed breakdown of all the major tasks and interactions between users and the system. In all the following specific use cases, the actor is the User.

1. **User Registration and Login**
   The user creates a profile, or logs into the system. (Figure 3.1)

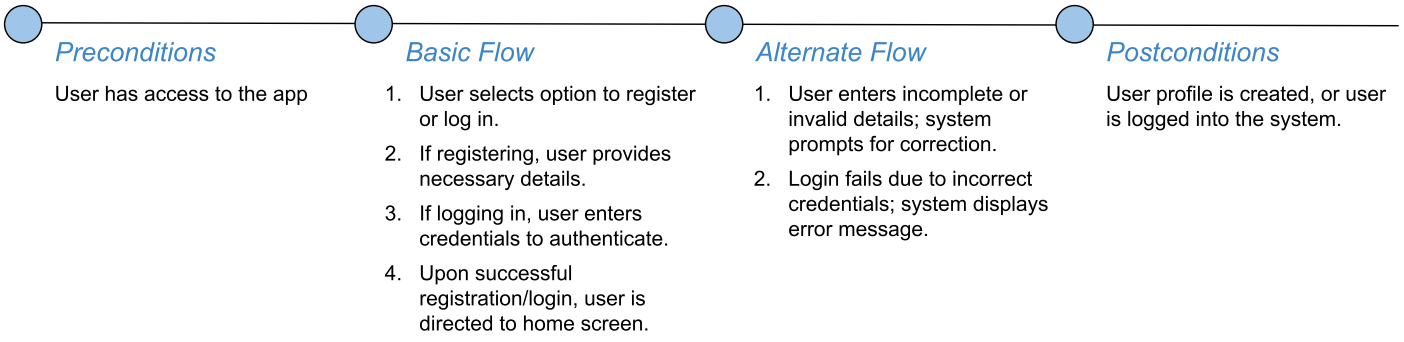


Figure 3.1: *User Registration Use Case Diagram*

2. **Content Sharing**
   The user shares content from a social media platform into the app. (Figure 3.2)

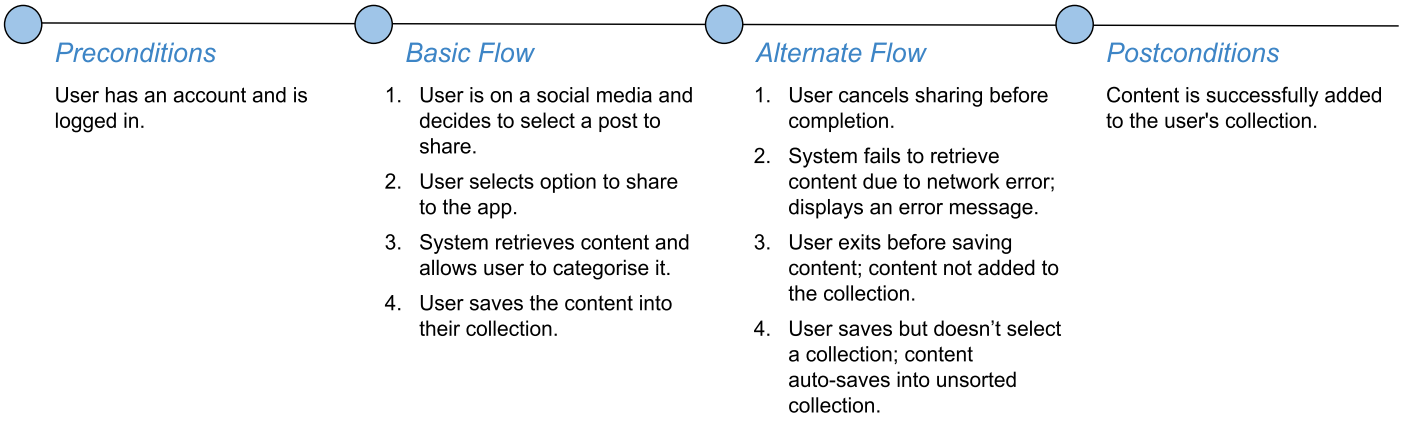


Figure 3.2: *Content Sharing Use Case Diagram*

3. **Content Categorisation**
   The user categorises saved content into specific collections. (Figure 3.3)

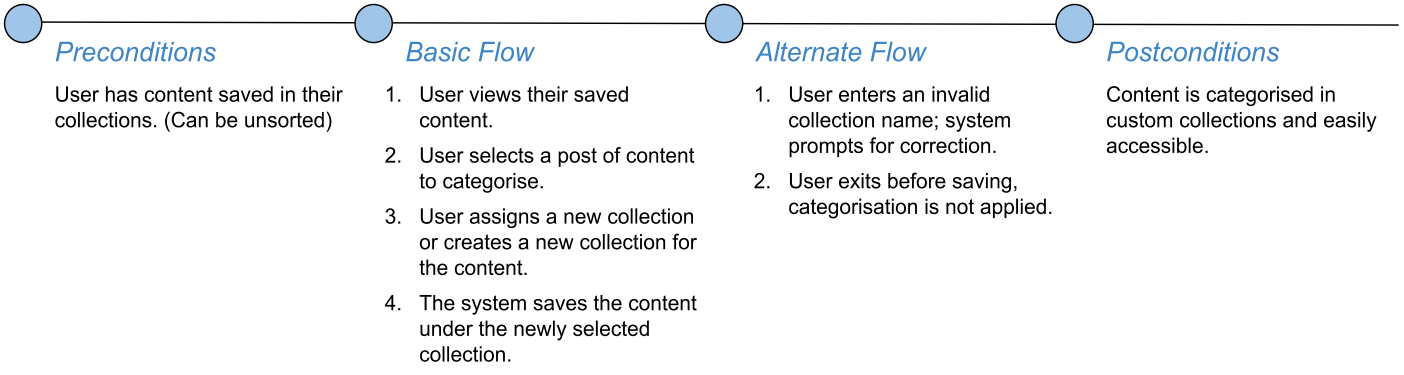


Figure 3.3: *Categorising Use Case Diagram*

4. **Content Search**
The user searches for specific content within their collection. (Figure 3.4)

Preconditions	*Basic Flow*	*Alternate Flow*	*Postconditions*
User has content saved in their collections. (Can be unsorted)	1. User enters a search query (e.g. keyword or by tag). 2. System retrieves matching results from users collection. 3. User browsers and selects desired content from the search results.	1. No results match the query; system displays this feedback.	User views relevant content based on their search query

Figure 3.4: *Search Use Case Diagram*

5. **Manual Post Creation**
The user manually adds a post by uploading an image or pasting a link. (Figure 3.5)

Preconditions	*Basic Flow*	*Alternate Flow*	*Postconditions*
User is logged in.	1. User presses the add button and selects to 'Add Post' from either the Home page or the Collection page. 2. User uploads an image or enters a link 3. User adds a title, description and tags 4. The system saves the post to the selected collection (or to the newly created collection)	1. User uploads an unsupported file; system prompts for a valid format. 2. Link is invalid or not supported; system prompts for verifying or changing the link.	Content is successfully saved as a new post in the desired (or unsorted) collection.

Figure 3.5: *Manual Post Creation Use Case Diagram*

6. **Post Editing and Deletion**
The user edits or deletes an existing post. (Figure 3.6)

Preconditions	*Basic Flow*	*Alternate Flow*	*Postconditions*
User has at least one saved post.	1. User selects a post from their collection. 2. User chooses to edit or delete the post. 3. If editing, the user modifies the details and saves, which is then altered and updated in the database. 4. If deleting, the system confirms the action and removes the post from the database.	1. User exits before saving changes; original post then remains unchanged. 2. User cancels deletion before confirm; the post remains.	Post is successfully updated or removed.

Figure 3.6: *Editing and Deletion Use Case Diagram*

7. Collection Management

The user creates, renames, or deletes a collection. (Figure 3.7)

Preconditions	*Basic Flow*	*Alternate Flow*	*Postconditions*
User is logged in.	1. User presses the add button and selects to 'Add Collection' from either the Home page or the Collection page. 2. User enters a name and description and confirms. 3. The system created the collection and adds to database. 4. For renaming or deleting, the user clicks on the collection and selects to modify or delete it.	1. User enters an invalid name; system prompts for correction 2. User cancels renaming or deletion before confirmation; collection remains unchanged.	Collection is created, renamed or deleted.

Figure 3.7: *Collection Management Use Case Diagram*

8. Viewing Post Details

The user views the full details of a saved post. (Figure 3.8)

Preconditions	*Basic Flow*	*Alternate Flow*	*Postconditions*
User has at least one saved post in a collection.	1. User navigates to a collection and selects a post. 2. System displays the full content of the post include the image (or embed), description and tags. 3. The user can interact (e.g. open in original platform)	1. System fails to retrieve details due to network or database connection issue; system displays an error message	Post and its full details are displayed.

Figure 3.8: *Viewing Posts Use Case Diagram*

9. Profile Management

The user views and edits their profile details. (Figure 3.9)

Preconditions	*Basic Flow*	*Alternate Flow*	*Postconditions*
User is logged in.	1. User accesses their profile in the collections page. 2. User clicks the button to edit their profile. 3. User updates details like name, profile picture. 4. System saves the changes and sends to database.	1. User enters invalid input (e.g unsupported image format for profile picture); system prompts for correction. 2. System fails to save the change due to a network or database connection issue; system displays error message.	Profile is successfully updated.

Figure 3.9: *Profile Managing Use Case Diagram*

10. User logout

The user logs out of the system. (Figure 3.10)

Preconditions	*Basic Flow*	*Alternate Flow*	*Postconditions*
User is logged in.	1. User navigates to the settings tab. 2. User selects the logout button. 3. System logs user out and returns them to the login screen.	1. System fails to log out due to a network issue; an error message is displayed.	User is logged out and must log in again to access their account and saved content data.

Figure 3.10: *Logout State Use Case Diagram*

11. External Link Handling

The user clicks on an external link (e.g the image link they saved or an embed link from their collection posts). (Figure 3.11)

Preconditions	*Basic Flow*	*Alternate Flow*	*Postconditions*
User has saved posts that contain external links or embeds.	1. User selects a saved post from a collection. 2. System displays the post details, including the external link redirector. 3. User clicks on the post embed image or the direct link. 4. System opens link in default browser or redirecting to platform (e.g instagram embeds)	1. External platform removes or restricts access to the content; system displays an error 2. System fails to open link due to a network issue; error message is shown.	User successfully views external content through redirection.

Figure 3.11: *External Link Handling Use Case Diagram*

5.2 System Architecture Design

The system follows a client-server architecture model (Figure 4) (see Appendix C for full-resolution diagram).

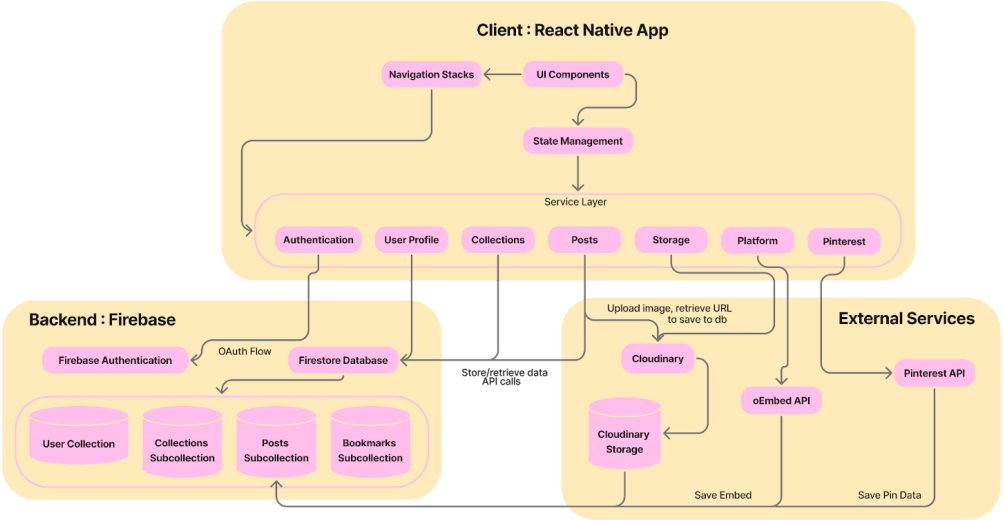


Figure 4: *Client-Server Model*

The architecture is divided into multiple layers for ensuring modularity, scalability and maintainability. The client (frontend) is a React Native mobile application, serving as the primary interface for user interaction. The backend is serverless and largely powered by Firebase. The application also interacts with external services such as Cloudinary, Pinterest API and oEmbed APIs. Asynchronous operations and Firestore real-time listeners are used for responsive UI updates.

5.2.1 Data Flow

The system ensures smooth data flow (Figure 5)

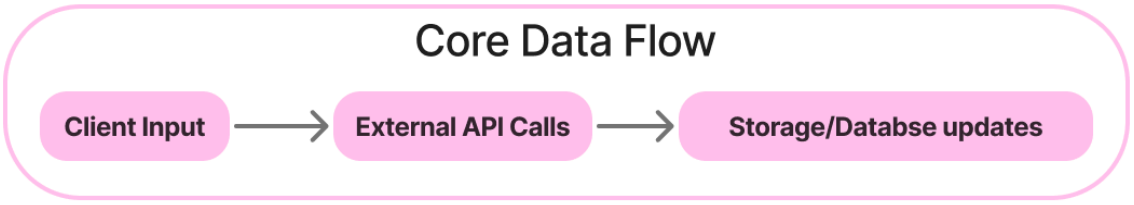


Figure 5: Data Flow Diagram

5.3 Database and Data Design

Firebase's Firestore is used as the primary database, employing a NoSQL document-based architecture with optimisations for mobile applications. The data design follows hierarchical patterns, strategically using document references and subcollections in the management of user-specific content.

5.3.1 Database Schema

The database is structured as shown below in Figure 6.

```
users/{userId}/collections/{collectionId}/posts/{postId}
```

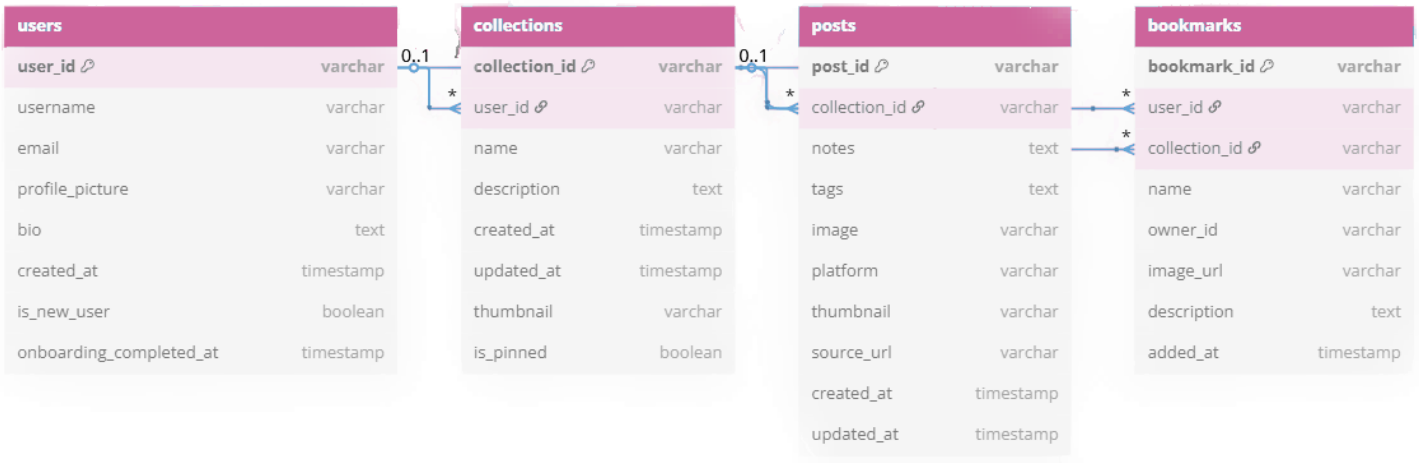


Figure 6: Database Schema Diagram

5.3.2 Data Organisation

User profiles are stored as documents in the top-level collection, with each document identified by the user's Firebase Authentication UID. This ensures direct mapping between authentication and user data as well as efficient access to user-specific information with security rules enforcements based on document ownership. The hierarchical layout under collections is designed for logical grouping and modular access, aligning with Firestore best practices and minimising reads, meanwhile allowing deep linking into individual collections or posts.

5.4 Content Organisation and Tagging

Content is organised within the app via two main structures: Collections and Tags. 'Collections' serve as primary containers, allowing users to group related content, whereas 'Tags' functions as cross-collection user-defined metadata, providing flexibility and allowing users the ability of content classification.

5.4.1 Manual Tagging

Tags are stored as arrays within post documents for both simplicity and performance, enabling efficient querying via Firestore's `array-contains` operators. Tag input is processed to handle comma separation and trim whitespace (Figure 7) (A).

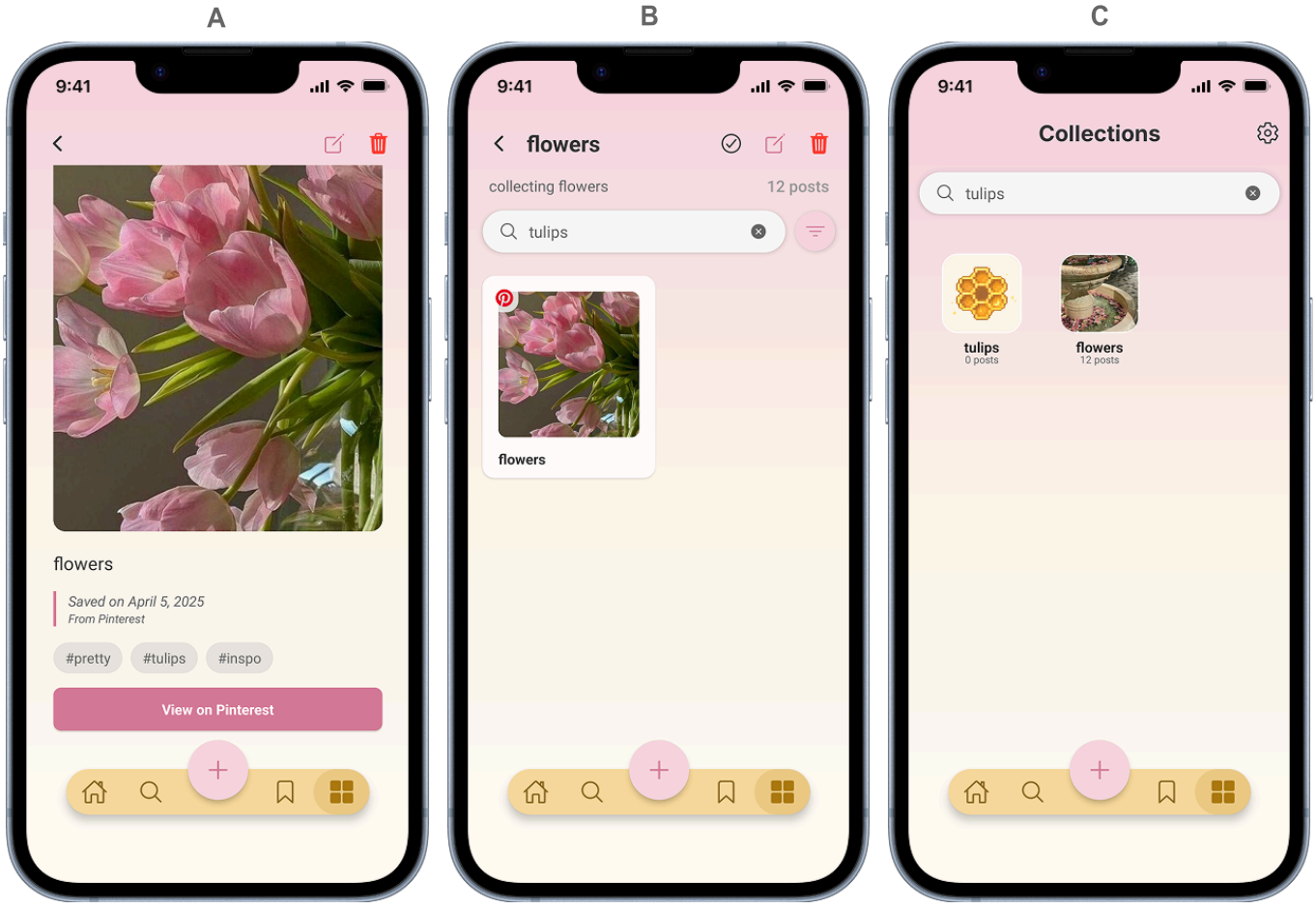


Figure 7: *Manual Tagging Features of 'Collecti'*

Within a collection, users can search for posts using respective tags (B). In the tab that displays all collections, users can also include the tags within search, displaying collections with relevant titles first, followed by the collections with posts containing the tag searched (C).

5.4.2 Content Recommendations

Tags play a key role in the recommendation engine described in Chapter 6.6, aiding in tag pattern analysis, using Natural Language Processing (NLP) for enhancing content discovery via the Homepage.

5.5 UI/UX Design

5.5.1 Branding

'Collecti' was chosen to reflect the core functions - organising content into collections. The name is memorable yet simple, conveying the essence of the app's purpose. The honeycomb/bee motif is incorporated as a thematic symbol.The symbolism includes bees as natural collectors of pollen organising what they have collected efficiently within the honeycomb to create honey as the end-product. Likewise, the functionality of the app will allow users to gather posts, categorise them and store them within structured collections.

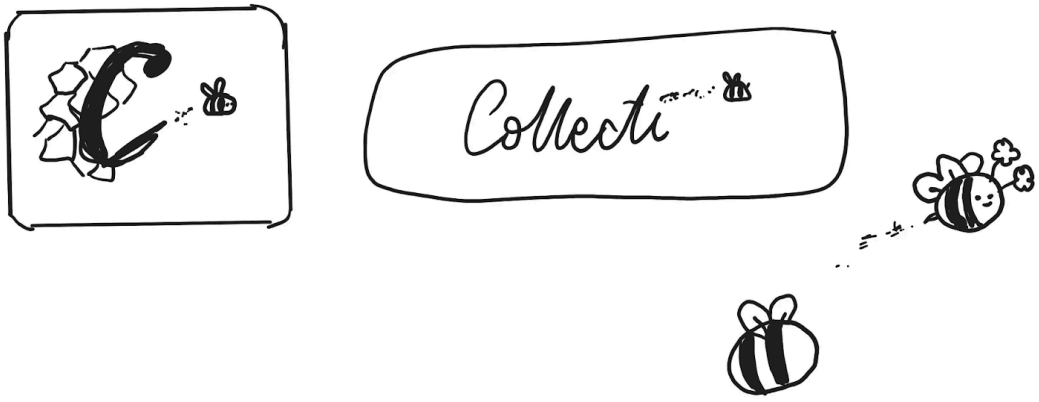


Figure 8.1: *Initial branding sketches*

Figure 8.2: *Actual Implemented Branding*

5.5.2 User Experience Rationale

The User Experience (UX) design of 'Collecti' was grounded in the core principles aimed for ensuring usability, clarity and ease of navigation for users - minimising friction and cognitive effort while enabling intuitive interactions. Drawing from foundational theories in human-computer interaction (HCI), heuristic principles and Norman's design principles, the interface was structured to support exploration, but also aiding error recovery and task efficiency.

Visual Hierarchy: The app employs visual hierarchy clearly, guiding users by emphasising key elements (e.g headings, buttons), presenting familiarity so that users can naturally focus on important actions, reducing confusion for first-time users (Lidwell, Holden, & Butler, 2010). This allows reducing extraneous cognitive load, grouping elements logically and supporting the user to focus on the task at hand, preventing breakdown (Heidegger, 1927*)* and disruptions of user flow. (Csíkszentmihályi, 1990).

Affordances: Affordances were leveraged in the indication of interaction, making sure buttons and clickable items are recognisable (Norman, 1988). Visual cues were utilised, helping users gain clarity about functionality.

Consistency: Across screens and features, consistency was implemented to enhance usability and reduce cognitive load. Navigation bars, input fields, search bars etc. follow consistent patterns across the app, maintaining predictability to enhance user experience.

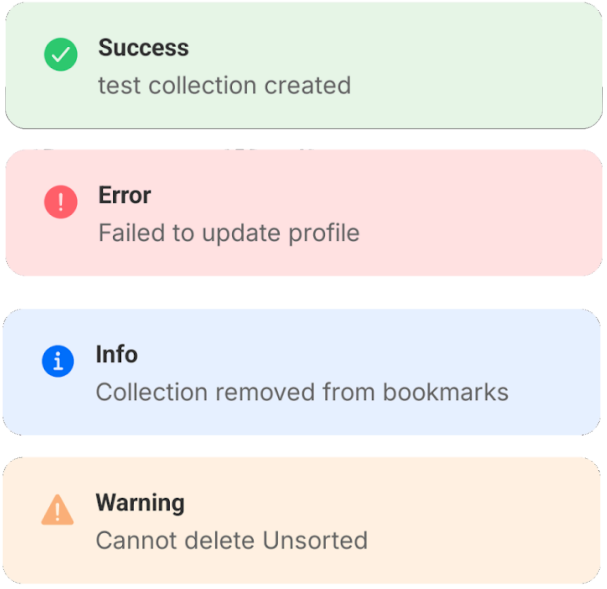


Figure 9: Toast notifications

Feedback: Mechanisms for clear feedback were provided - toast notifications, button animations, loading indicators etc. ensuring that users know the results of certain actions. This is aided further with error prevention and recovery, which are affordances included in the design structure. Figure 9 shows an example of the toast notification system, allowing for users to receive real-time validation of inputs and actions. Toasts are colour-coded to ensure familiarity with the outputs given.

5.5.3 Onboarding Screens

The onboarding screens (Figure 10) were designed to help users understand the core functions of the app before beginning to interact with it - being displayed right after signing up. A brief overview is implemented, explaining the app's purpose and how to use it. (A)

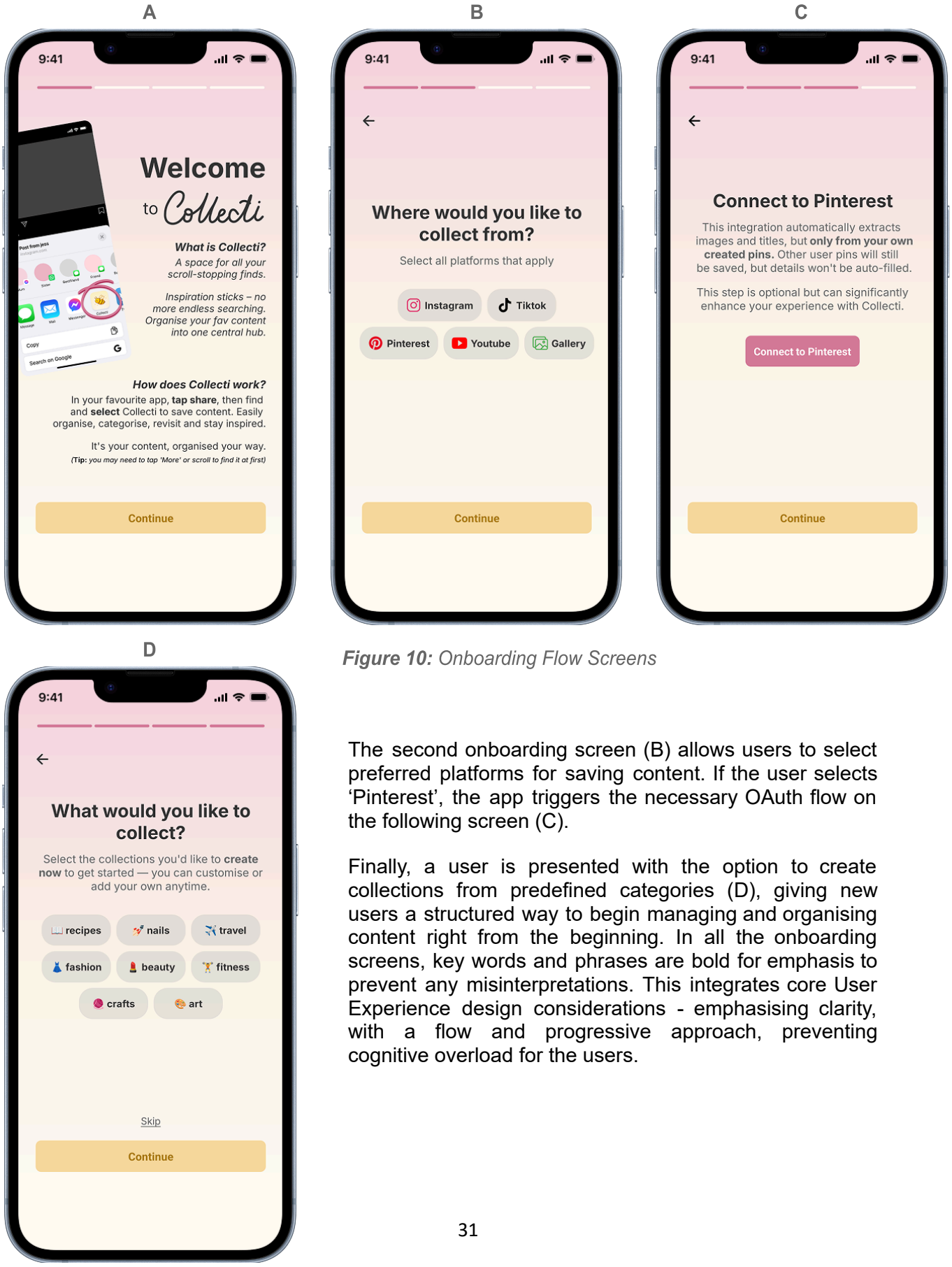


Figure 10: *Onboarding Flow Screens*

The second onboarding screen (B) allows users to select preferred platforms for saving content. If the user selects 'Pinterest', the app triggers the necessary OAuth flow on the following screen (C).

Finally, a user is presented with the option to create collections from predefined categories (D), giving new users a structured way to begin managing and organising content right from the beginning. In all the onboarding screens, key words and phrases are bold for emphasis to prevent any misinterpretations. This integrates core User Experience design considerations - emphasising clarity, with a flow and progressive approach, preventing cognitive overload for the users.

5.5.4 Wireframing and Prototyping vs. Final Design

Early wireframes, user flows and screen mockups were created to visually plan and define the application structure and layout before proceeding with development. Several iterations were made between the prototype and final products, adjusting UI elements and tweaking for streamlining usage and intuition in app usage. Examples can be seen below (Figure 11.1, Figure 11.2) of the development of the two initial screens a user sees before delving further into the app.

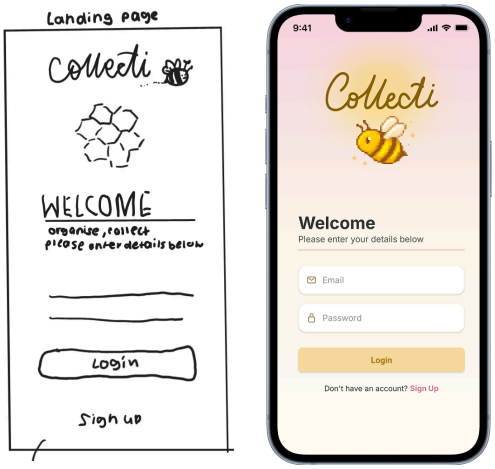


Figure 11.1: *Login Wireframe vs Final*

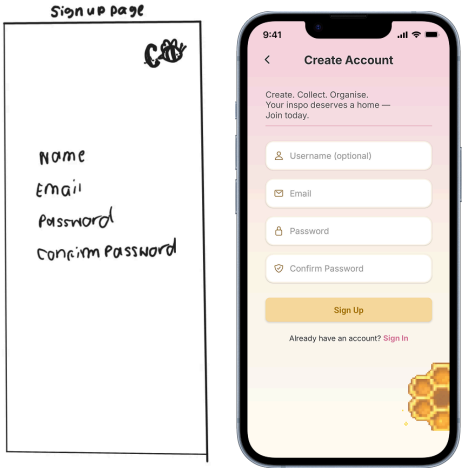


Figure 11.2: *Sign Up Wireframe vs Final*

Chapter 6: **Implementation**

6.1 Technology stack

In table 3, the technology stack used for implementation is outlined.

System Requirements	*Version*	*Purpose*
React Native	v18.2.0	Cross-platform mobile development
Expo	SDK50	Build toolchain and services
Firebase	v9.0.0	Authentication + NoSQL cloud database
Cloudinary	v2.6.0	User-uploaded media management
JavaScript	ES6+	Core programming language
Node.js	v14+	Development environment

Table 3: *Final Tech Stack/System Requirements of the app*

6.2 Mobile Development Environment

During the initial phases of planning and development, it was unclear whether a mobile-based application or a web application was most appropriate. Given that the primary use case for the app revolves around social media content, which is primarily consumed on mobile devices, it was clear that a mobile-first approach would be the most efficient solution. It leverages the device that users are already accustomed to, without needing to navigate through browser interfaces. The app appears in the native share sheet/intent, pointing to a more direct integration.

The initial prototype employed Expo Go for the development environment (Expo, 2024), which simplified testing of React Native applications on mobile devices, without the need for app compiling, providing a fast and convenient way to test features in real time. It was a great tool for initial development and quick front-end testing, however there were great limitations that would impede the integration of certain key features. The most significant issue was restricting motivations to native modules, meaning certain necessary libraries such as `expo-share-intent` (AChorein, 2024), which enabled content sharing between apps couldn't be implemented. These libraries required changes to XML files within the Android project, therefore being unsuitable for Expo Go's managed workflow. This meant that the sharing features, which were essential for the app's functionality, could not be fully accessed this way.

A solution was switching from Expo Go and learning documentation to begin a custom-managed Expo and React Native project (React Native, 2024). This enabled full access to the necessary native modules to make required changes to integrate the share intent functionality.

6.3 Authentication Implementation

6.3.1 Authentication Service

For authentication, Descope was initially considered due to the promoted streamlining of OAuth2 setup and client SDKs in managing sessions and custom login interfaces. However, after further evaluation, switching to Firebase was established as the most viable option. While Descope offered an intuitive interface, there were limitations during the initial integration, particularly in the adaptability of client SDKs to a React Native setup and reduced compatibility with third-party libraries necessary for other app features.

Firebase Authentication, being part of the larger Firebase platform, allowed seamless integration of authentication services in conjunction with the database, streamlining backend development flow.Its extensive documentation further facilitated the development process, compared to Descope. With built-in support for multiple authentication methods, the flexibility aligned perfectly with the application requirement, helping to streamline access. Furthermore, the offered free tier was better aligned with the project budget.

6.3.2 OAuth Integrations

User Authentication was intentionally kept simple for the scope of this project, relying solely on email and password login through Firebase. Initially, incorporating extra OAuth logins was planned, however, there were a variety of practical concerns. Many of the supported sharing platforms did not have directly accessible OAuth possibilities. Google's OAuth service was also considered, but upon further investigation, access to this service required a $25 developer registration. This was deemed a non-essential expense and was deferred as an implementation.

6.3.3 Session Management

User sessions were managed through Firebase, implementing persistent login and real-time auth state handling to ensure a seamless and secure user experience across app launches. Persistence is created through using AsyncStorage to maintain authentication across user sessions on app restart. Once a user logs in, a session token is stored securely, allowing them to remain authenticated unless they explicitly log out or the session expires:

```
auth.setPersistence(firebase.auth.Auth.Persistence.LOCAL)
```

In mobile environments, this is essential as users expect continuity without frequent interruptions or disruptions of flow (Csikszentmihalyi, 1990).

Authentication logic was encapsulated as a dedicated service model (`authService.js`), isolating it from UI components to promote modularity. The root (`App.js`) implemented an authentication state listener, using Firebase's `onAuthStateChanged`, enabling reactive UI updates based on authentication status. If a user is authenticated, they are directed to the main app flow. The system also accommodates additional tokens such as Pinterest OAuth, scoped to each Firebase user, ensuring third-party integrations are tied to authentication sessions without security compromise.

6.4 Social Media API Integrations

6.4.1 Instagram

Instagram Basic Display API seemingly provided all the basic limited functionalities perfect for integration, however due to its deprecation, it was concluded to be unsuitable. A majority of the available information and documentation was slightly misleading perhaps due to being outdated. The main API offered (Graph API) is restricted to usage by business and creator accounts, mandating app review and business verification. Instagram offered a direct oEmbed API too, yet it also required an Advanced Access approval and a publicly available app, conflicting with the prototype/limited-dev stage nature of the project.

Third-party solutions like Instagram Scraper APIs (RapidAPI) were explored. It functioned well in terms of metadata extraction and customisation, allowing the desired fine control over display and styling. However, the API had significant constraints, only allowing 30 requests/month, with a subscription cost of $20/month for more requests. This was not feasible for a small-scale project like 'Collecti'. There is also a risk of non-compliance with the Terms of Services imposed by Scraper APIs.

To omit these limitations, instead of subscribing to an API, the built-in Instagram oEmbed API was utilised, making use of public post URLs for displaying content directly in-app (Figure 12). Embedded posts maintain Instagram's native layout, without options to tailor visual presentation or functionality. There is also no metadata retrieval so captions cannot be displayed.

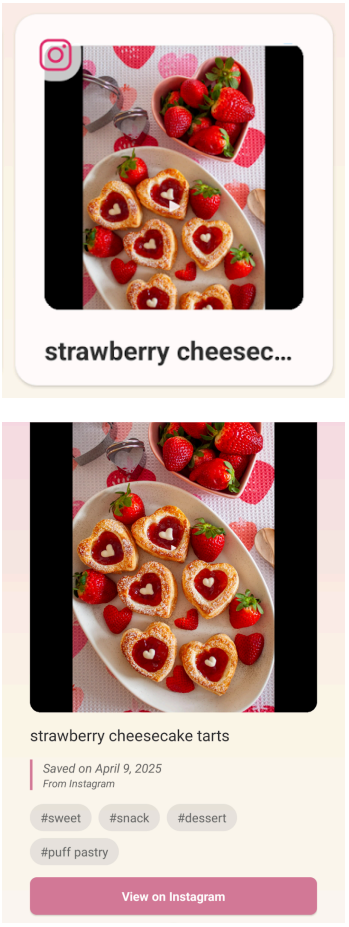
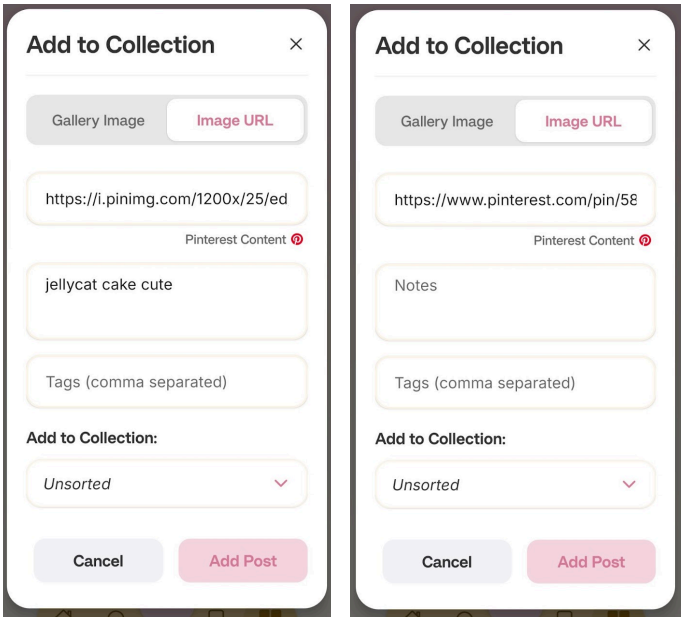


Figure 12: *Example Embed Post Thumbnail*

Despite these drawbacks, the embed satisfies the core project requirements while adhering to budget and technical constraints. Using an embed also means a user has direct redirection access back to the original saved content. For future development, obtaining required approval or incorporating paid third-party APIs could be a viable solution to enhance functionality and a more polished user experience.

6.4.2 Pinterest

For Pinterest, short-lived generated access tokens were used for the experimental phase of development, however, due to their short-lived nature and errors, they were not implemented. Instead, access was secured through the Trial Access API, which marked a critical milestone by enabling core functionality for retrieving a user's Pinterest content. For this API, an OAuth 2.0 verification flow had to be integrated, as mandated by Pinterest for secure and authorised access to user data. This standard protocol ensures that users grant explicit permission for data sharing without exposing their credentials. The integration process was supported by the official Pinterest tutorial (Pinterest, 2024b) and Developer API Quickstart Guide (Pinterest, 2024c) which assisted in configuring the redirect flow, token exchange and scope handling within the app. The success of this flow enabled users to connect Pinterest accounts to 'Collecti' and import their own content.



However, during development and testing, a significant limitation was discovered: the Pinterest API only allows access to a user's *own content* through 'GET' API calls. As a result, direct metadata extraction through the API was not possible for pins created by other users. To address this, a hybrid fallback system was implemented, where if a user has been authenticated and OAuth-verified, their personal pins can be fetched via the Pinterest API with metadata being extracted, meaning the modal for adding posts could automatically be pre-filled. (Figure 13)

Figure 13: *Pinterest API (left) vs. Pinterest oEmbed (right)*

If a pin doesn't belong to the user, the app will resort to utilising the oEmbed API, enabling embedding and displaying public pin content visually. While this limited metadata retrieval depth, it ensured visual consistency and allowed external pins to still be saved and browsable within the app. (Figure 13) The embed and API content retrieval is displayed similarly across any post regardless of platform, similar to Figure 12.

6.4.3 TikTok

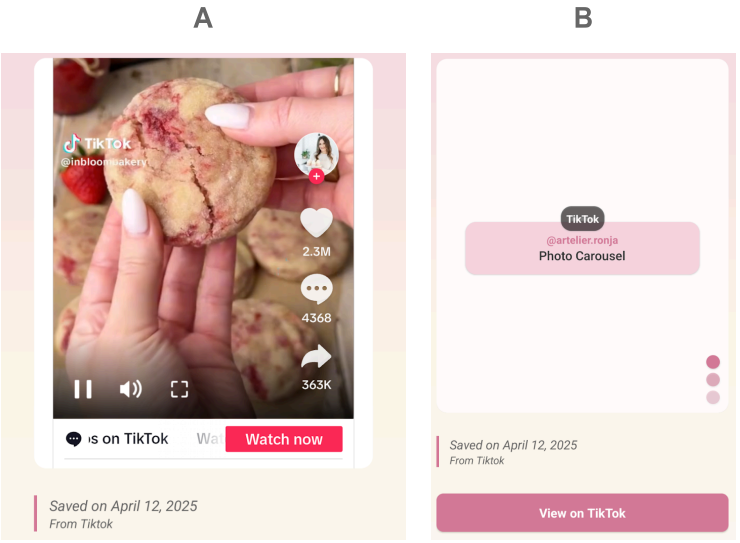


Figure 14: *Tiktok Embedded Post and Carousel Fallback*

As aforementioned in Chapter 2, there is no official available public API for TikTok that supports the retrieval of post metadata or content in the manner required, specifically, the programmatic extraction of data such as video URLs, captions, and thumbnails from arbitrary public posts.

To address this, the app leveraged the oEmbed API to allow embedding of public video content (Figure 14) directly into the app using the shareable URLs with in-app playback support (A). As of now, the oEmbed API does not currently support embedding of carousel-style posts (e.g. singular or multiple images), limiting the type of content possible to display. Due to this, a fallback screen was implemented (B), meaning the user could still access the content via the navigator button.

6.4.4 Youtube

YouTube also presented similar API limitations as the previously mentioned platforms. Although a Data API exists, key registration and strict quote management were required, which was deemed excessive for the project scope. For the final implementation, YouTube content made use of the oEmbed API allowing videos to be displayed and played directly

within the application interface in the post details screen. This is similar to the functionality of TikTok embeds, as shown in Figure 14.

6.4.5 oEmbed

To fit embeds smoothly into the app, a custom HTML wrapper view component was developed to dynamically load the embed content, ensuring responsiveness and correct aspect ratios. This gave embeds a more native integration feel instead of simple iframes. Like with all embeds in this scope, a key challenge was the automatic navigation away from the app upon clicking embeds. Thumbnails for collections are the most recently added posts, meaning embeds could be used for this visual. The redirective behaviour in this scenario introduced friction from both a UX and content control standpoint, disrupting user's flow and potentially leading to unintended navigation away from the app. To mitigate this, a click-blocking overlay was implemented to prevent accidental redirects. Each post detail view featured an accessible "View on (platform)" button, (Figure 15) providing a clear and intentional way for users to still be able to view the original content externally, maintaining user control and interaction flow within the app.

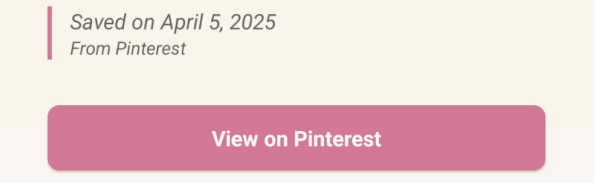
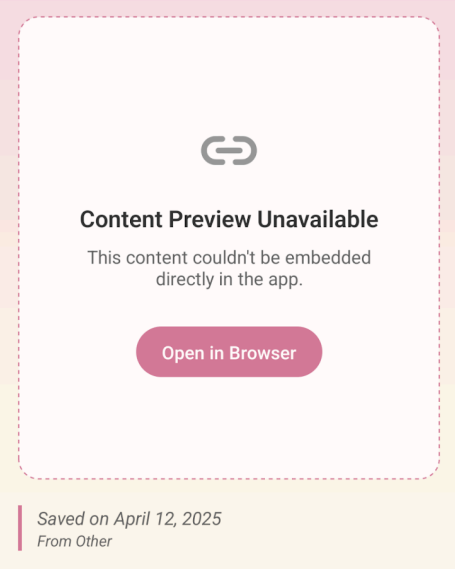


Figure 15: *Platform Navigator Button*

6.4.6 Error Fallback



If a Web or image URL is shared from an unsupported platform, or it is met with invalid format or other errors, the app gracefully displays a fallback screen (Figure 16), informing the user that a preview of the saved content is not available within the Collecti app, with a redirecting 'Open in Browser' navigator button as an alternative.

To further maintain visual consistency, unsupported posts default to a placeholder thumbnail in collection overviews, aligning with resilient UI principles.This ensures users are not met with blank or broken views (Norman, 1988). Through preventing affordances and incorporating fallblack flows, the app is able to reduce friction and prevent dead ends, even in edge-case scenarios.

Figure 16: *Error Fallback Post Details*

6.5 Storage System

The application uses a hybrid approach for data and media storage, utilising both Firebase Firestore for structured data and Cloudinary for storing manual media assets like user-uploaded images to enable scalability and cross-device media accessibility.

6.5.1 Firestore

Firestore serves as the primary database for structured content storage such as profiles, collections and posts. (Chapter 5.3) When a user saves a post, metadata is contained and references to any associated media are stored as URLs. Cloudinary supports this process.

6.5.2 Cloudinary

Cloudinary is used for storing manual user-uploaded media assets, both in the scope of saved content, as well as custom profile-picture handling. As a cloud-based media CDN, it ensures efficient retrieval and rendering across devices. For implementation, the official SDK was used with serverless upload functions configured to handle authenticated uploads securely. When an image is uploaded, it is compressed client-side before being uploaded via a signed POST request. Upon completion, a secure URL is returned and saved into the corresponding user document in Firestore for future referencing and rendering. Offloading image storage from the device or local database reduced read costs, keeping the app lightweight and ensuring we can still use the free Firebase plan.

6.6 Recommendation System

6.6.1 Approach

A content-based recommendation system was implemented leveraging Natural Language Processing (NLP) techniques for keyword extraction and semantic similarity scoring. User-generated content such as post descriptions, tags, titles, was processed into vector representations, which enabled the generation of recommendations based on textual relevance to a user's existing collections. (Figure 17).

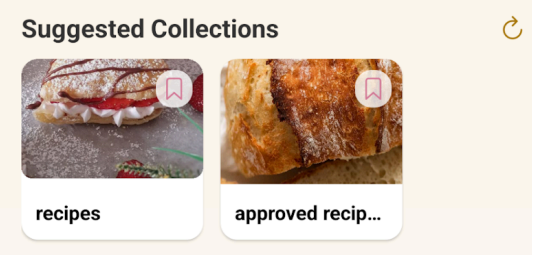


Figure 17: *Recommendations*

The vector space model is applied to compute similarity between items, offering personalised suggestions based on overlapping content themes. If there is insufficient data, the system falls back to a popularity-based model, where collections are sorted by creation timestamp (`createdAt`) in descending order. This fallback is under the assumption that recency can be used as a proxy for relevance, surfacing only the most recently created collections from across all users in the database via Firebase's `collectionGroup` query. Content-based filtering with recency-aware fallback allows for an interpretable and scalable recommendation system, maintaining computational efficiency (Lops et al., 2011).

6.6.2 Future Enhancements

Although the fallback model provides interpretable results, it is not sensitive to user interaction or contextual intent. Evaluation of the recommendation system revealed two main shortcomings: There is no engagement feedback loop, meaning without tracking user clicks or re-visits (engagement), recommendations can remain static even if relevance varies over time. There is also topic overfitting, meaning content can be over-represented in a user's early use. This skews suggestions and potentially narrows discovery with bias towards over-represented categories. Currently, the system prioritises simplicity over personalisation depth, therefore future work should involve hybrid learning techniques that combine implicit feedback (e.g. views, engagement) and dynamic weighting based on user session patterns, to allow for contextual awareness.

6.7 Sharing Content to Collecti

6.7.1 Supported App flow

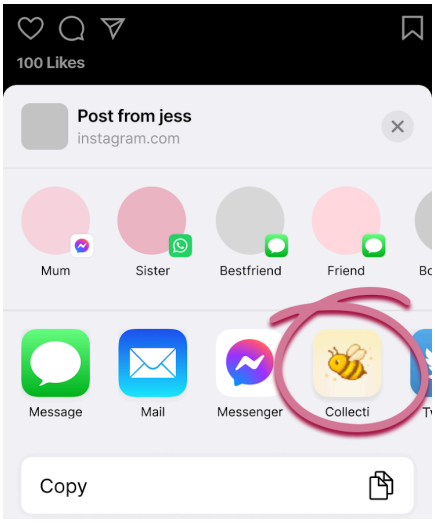


Figure 18: Share Sheet

Collecti integrates content-sharing seamlessly using React Native's share intent library and platform-specific detection logic. If a user shares a post from a supported platform, the URL via the share intent is intercepted and the platform is automatically detected using URL pattern recognition. This triggers the post creation modal with pre-filled data where possible (Figure 13). This flow is also introduced in the onboarding process, helping users understand how to share content directly into the app. Figure 18 demonstrates how the app appears in the native share sheet.

For API supported apps like Pinterest, metadata such as images or titles can be enriched via backend calls. Embedded rendering is handled using dedicated embed components such as `PinterestEmbed.js` //`<PinterestEmbed>` . Automation in this sense ensures minimal friction by aligning with user expectations for immediacy and continuity within mobile applications.

6.7.2 Unsupported App flow

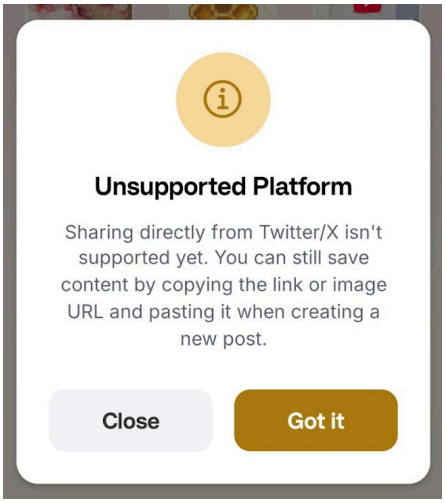


Figure 19: Unsupported Modal

If the user attempts to share from unsupported platforms, fallback is implemented for this scenario. Unknown sources are detected and presented with a modal explaining that direct sharing isn't available, offering alternative entry methods. (Figure 19) Unsupported posts can still be saved as presented in Figure 16. In each of these cases, there is a clear call to action, following Norman's principles about resilience and feedback. (Norman, 1988).

Chapter 7: **Presentation**

7.1 Codebase Structure

Collecti is built with a modular and scalable architecture, designed to support maintainability with clear separation of concerns.

7.1.1 Codebase Folder Organisation

`src/` : Contains core application source code.

- `components/` : Reusable UI components
  - `embeds/` : Platform-specific embeds for rendering external media content in-app
  - `layout/` : Layout scaffolding components for structuring screens
  - `modals/` : Modal dialogs and overlays
  - `utilities/` : Stateless UI primitives and helper components
- `hooks/` : Custom React hooks for reusable state/logic
- `images/` : Static image assets
- `navigation/` : Defines app-wide navigation logic and screen flow
  - `stacks/` : Stack-based navigator configuration
- `screens/` : Screen-level components corresponding to app pages/features.
  - `Bookmarks/` : Bookmark viewing and management
  - `Collections/` : Organising user collections and posts + user settings
  - `HomePage/` : Primary landing screen with personalised content
  - `Search/` : Search Interface Screen
  - `SignIn/` : Login flow and auth UI
  - `SignUp/` : User registration flow and onboarding
- `services/` : Business logic and backend integrations
  - `pinterest/` : Pinterest API integration and data handling
- `styles/` : Shared styling modules and theme configurations
- `utils/` : General-purpose utility functions, constants, helpers

Each item in the codebase is commented, with a full item overview at the top of every file to outline the purpose of the code, including any patterns or principles followed. The code implements a clean architectural split between UI, logic and data access layers, following the separation of concerns principles. Additional screenshots of features not discussed in detail, such as the bookmarks, user settings, modals etc. are included in Appendix E.

7.1.2 Naming Conventions

Standard react native naming conventions are utilised for the Collecti app/coding, following a feature and function based folder structure. (Table 3)

Item	Convention	Examples
Components/Screens	`PascalCase`	`CollectionDetails.js`
Functions/Variables	`camelCase`	`fetchUserPosts()`
Hooks	`useCamelCase`	`useCollectionSeach()`
Files (non-component)	`camelCase`	`statUtils.js`
Constants	`UPPER_SNAKE_CASE`	`DEFAULT_LIMIT`

Table 4: Naming Conventions used

7.2 GitHub Repository

The source code with instructions can be found on GitHub:
https://github.com/jmalekx/collecti

7.3 External Libraries and APIs used

Table 4 outlines the External Libraries utilised for the implementation.

Library/API	Version	Purpose
React Navigation	v6.0.0	Navigation and routing
React Native WebView	v13.2.2	Embedding platform content
React Native Vector Icons	v9.2.0	UI icons (Ionicons, MaterialIcons, AntDesign, FontAwesome)
React Native Linear Gradient	v2.8.3	Background gradients and styling
React Native Toast Notifications	v3.3.1	User feedback system
Expo Image Picker	v14.3.2	Media selection from device
Expo Share Intent	v1.2.0	Deep linking and content sharing
Expo Google Fonts	v0.2.3	Typography (Inter font family)
AsyncStorage	v1.18.2	Local data persistence
Axios	v1.4.0	HTTP client for API requests

Library/API	Version	Purpose
base-64	v1.0.0	Encoding for Pinterest API authentication
Linkify-It	v4.0.0	URL detection in text content
Pinterest API	v5.0.0	Pinterest integration (OAuth, pin data)

Table 4: *External Libraries and APIs used*

Chapter 8: **Testing**

8.1 Unit and Integration Testing

Unit tests were conducted to check the functionality of various features of the Collecti app. As the app was being developed, every new component/screen/feature added was thoroughly tested. Tests were also executed using Jest. Table 6 shows the results of the key functionality tests.

Test Case	Steps	Expected Output	Pass
User signup	1. Open app 2. Click on sign up here 3. Enter details 4. Click register 5. Start onboarding	Account is created and added to database. User is navigated to onboarding to complete signup.	✓
Onboarding	1. After signup, navigated to onboarding 2. Select platforms 3. Connect if prompted 4. Select suggested collections or skip 5. Complete onboarding	If user selects Pinterest as platform, they are prompted to connect via OAuth for API access. If user selects suggested collections, they will be created upon completion. User is navigated to homepage.	✓
User login	1. Open app 2. Enter valid details 3. Click login	User is logged in and authenticated before being navigated to the inside layout of the app, the Homepage.	✓
Collection Creation	1. Login to app 2. Click add button 3. Select Add collection from menu 4. Enter collection name and optionally description 5. Click create	Collection is added and visible in the collections tab, with correct name and description.	✓
Viewing Collection	1. Login to app 2. Go to collections tab 3. Click on a collection	Collection details screen opens, showing all saved posts added to that collection.	✓
Deleting Collection	1. Login to app 2. Go to collections tab 3**a**. Long press a collection 4. Click delete icon 5. Confirm deletion 3**b**. Click on collection 4. Click delete icon 5. Confirm deletion	Collection disappears - it is deleted alongside the content within it.	✓

Test Case	Steps	Expected Output	Pass
Manually Create Post	1. Login to app 2. Click add button 3. Select add post 4. Upload image from gallery or paste an image url 5. Assign to a collection (or leave as unsorted) 6. Press Add Post	Post is created and appears in the chosen collection. Post is visible with correct image or link	✓
Share Post from Supported Platform	1. Login to app 2. Go to a social platform e.g. Instagram. 3. Find post you like 4. Click share 5. Select to share to Collecti app 3. Confirm create post.	Post appears in chosen collection. Post is saved correctly. Platform it was saved from is detected. ‘Add modal’ appears upon sharing to app. Link is auto pre-filled from the share.	✓
Share your OWN pin from Pinterest	1. Login to app 2. If not connected during onboarding, go to settings from collection tab and press connect to Pinterest. 3. Allow app access for OAuth verification 4. Share your own created pin to Collecti app.	If sharing a pin the user created on Pinterest (not other people’s pins - this is an API limitation), then it will be saved with image and metadata already filled out (post notes)	✓
Data Persistence	1. Login to app 2. Save some posts or create collections or save bookmarks 3. Close, restart app	Collections and data remain saved, user remains logged in (sessions)	✓

Table 6: *Unit Testing Outcomes*

8.2 User Testing

To assure realistic and unbiased feedback, unmoderated remote usability testing was conducted, allowing independent exploration of the app in the user’s own environment and devices, without direct guidance. This reflected real-world usage with higher accuracy, capturing authenticity and reducing pressure and influence that could be seen in live moderated tests. A small group of target users took part in the testing, being given a clear checklist of core tasks to complete inside the app. This ensured consistency across testers, and allowed exploration of the interface at own pace. Feedback was collected via a google form which participants filled out (Appendix D). Task completion checklists, usability Likert-scale ratings as well as open-ended questions for qualitative insights and improvement suggestions were included for user evaluations. The responses were recorded but remained anonymised. This allowed understanding of user interaction with the app, identifying usability issues and the level of intuition across the app for first-time users.

8.2.1 Statistical Results

Statistical Results were extracted from the user testing, meaning overall ratings of the app (Table 8) could be collected, and task completion success rates (Table 7).

Task Completion	
Total tasks attempted	47
Tasks completed successfully	44
Success Rate	93.6%

Table 7: *Task Completion results*

User Feedback	
Average usability rating	4.7/5
Visual appeal rating	5/5
Perceived usefulness rating	5/5

Table 8: *User Feedback results*

8.2.2 Positive Feedback

The majority of feedback received was positive, with users appreciating both functionality of the app, but also its design. Several key aspects stood out:

- **Ability to easily revisit saved content on external platforms:**
  The app demonstrated convenience and intuitive functionality.
  - *"Being able to go back to the exact post that was saved from your collection…is easy and swift to use."*
- **Aesthetic appeal:**
  The aesthetic qualities of the app were affirmed with positive reception of visual design.
  - *"Very aesthetic and pleasing to the eye"* with a *"very nice logo and concept behind the app".*
- **Non-social nature:**
  Users appreciated this nature, standing in contrast to the pressures and overload often found in social media platforms.
  - "*Not everything has to be a social space, and this helps us in this day and age of social media overload. It gives a sense of peace".*
- **Creative potential:**
  The app's value as a tool for organising inspiration was reinforced with its recognised creative potential.
  - *"Use of it as a creative inspiration outlet - having your inspiration in one place, it's like a virtual vision board!"*

- **Usefulness:**
  Several users mentioned the practicality and benefits of the app, aligning with the use case described in the project definition.
    - *“I am very into cooking, so being able to save recipes in one space really helps all the jumble-brained people like me.”*
    - *“Yes, I can save recipes in one place and not have to go to multiple apps.”*

8.2.3 Identified Issues and Insights

There was also a few friction points which emerged:

- **Sharing from platforms:** Users struggled to recognise the Collecti app in the native share sheet and expected more clarity in instruction.
- **Tag and collection search:** Tag search functionality was limited to collection details screen, therefore not being possible to search for tags in collections tab.
- **Onboarding confusion:** Automatic creation of collections when selecting interests led to misunderstandings due to lack of clarity in instruction.
- **Navigation affordances:** Users expected the user insights cards on the homepage to be interactive, but found them static.
- **Minor UI issues:** Some users noted visual errors like a clipped logo due to incorrect padding/responsiveness.

8.2.4 Improvements Based on Feedback

In response to the identified issues in feedback, the following improvements were implemented:

- **Onboarding clarity:** Instructional texts were adjusted for better clarity, ensuring users understood details about the app like how to share, what the Pinterest OAuth is for, and the automatic collection creation process. (Figure 20)

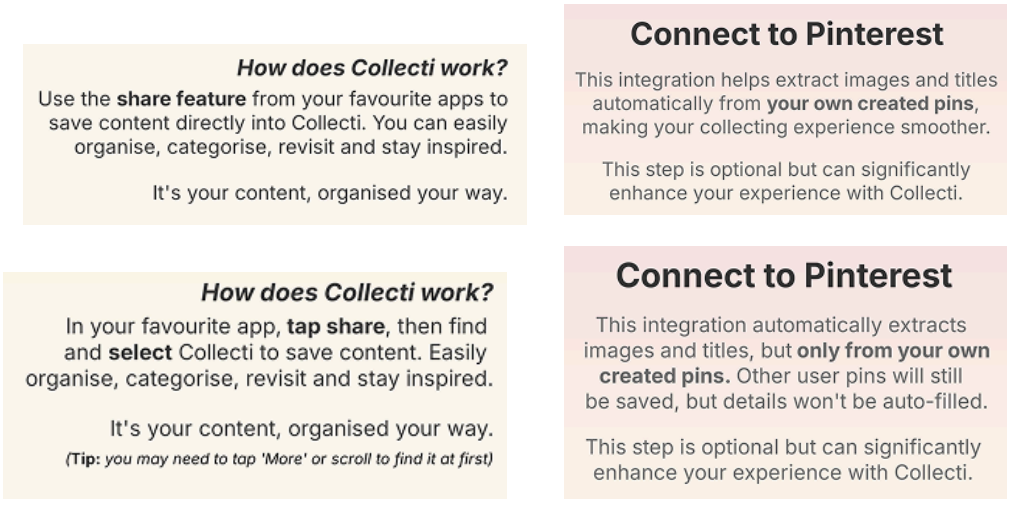


Figure 20: *Initial (Top) vs. Improved (Bottom) phrasing*

- **Interactive User Stat Cards:** The user stat cards on the homepage (Figure 21.1) were made interactive, allowing users to click through to the collections tab from the ‘collections’ and ‘posts’ card. A new screen was also implemented following a user suggestion to display expanded user analytics (Figure 21.2). This user had mentioned that such an implementation could make this part of the app *“more interesting”*, being *“not really a functional thing, just a cool to know’ thing”*.

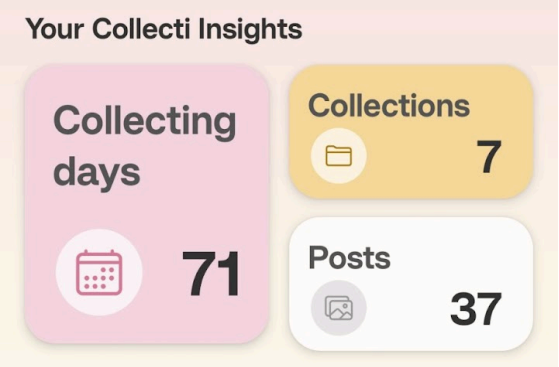


Figure 21.1: Statistic Cards

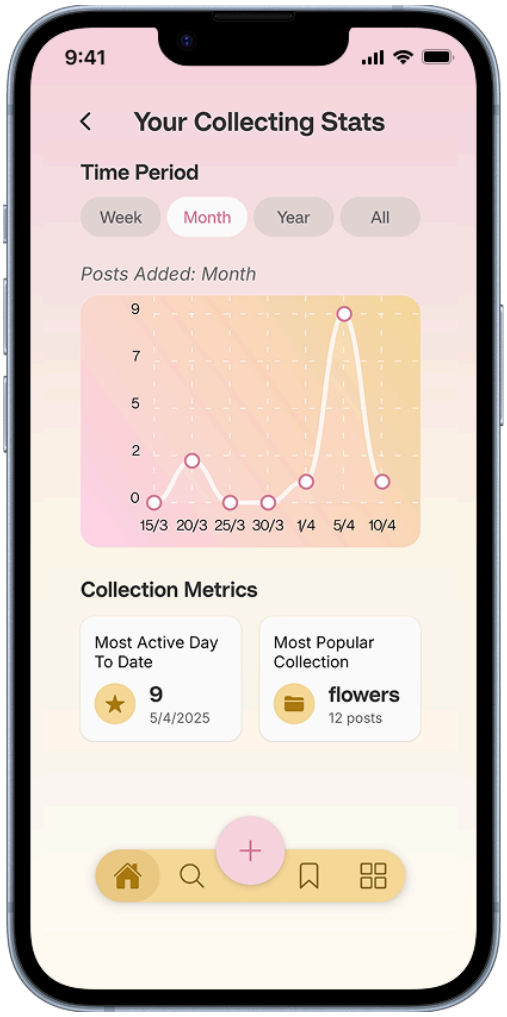


Figure 21.2: Overall Statistics Screen

- **Enhanced Tag Search:** The tag search functionality was expanded to allow ‘tag’ searches from the collections tab and not just from within a singular collection. Firestore’s full-text indexing is used to prioritise titles and tags over raw descriptions when searching.
- **UI Adjustments:** Any minor display issues such as the logo cutoff were addressed, ensuring correct responsiveness across different devices.This was tested through the use of the Emulator in Android Studio.

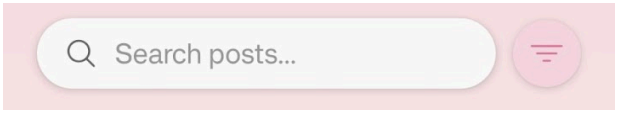


Figure 22.1: Added Sorting Button

- **Visual Sorting:** Based on a feature suggestion in the feedback, a user highlighted the need for more control over how saved content is accessed and visualised. A sorting option was implemented, allowing to sort saved content within a collection, based on date added and by name, streamlining retrieval of specific content.

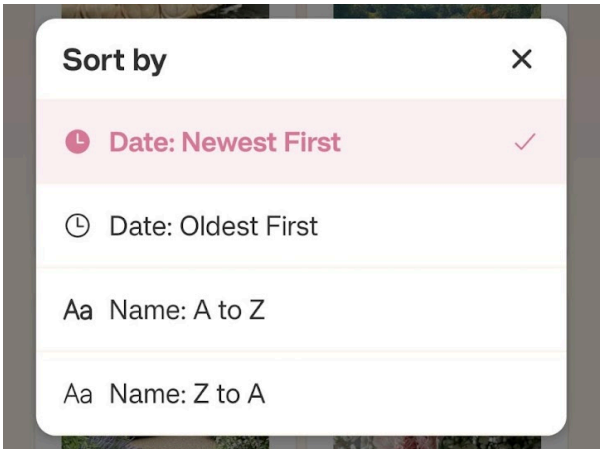


Figure 22.2: Sorting Options

Chapter 9: **Evaluation**

9.1 Achievements

The project aimed to design and implement a mobile application that enables users to save, organise, and revisit content from multiple social media platforms in one unified space. All core functional and non-functional requirements were fulfilled, as documented and validated through the conducted testing in Chapter 8. Crucial features of the app successfully allow users to:

- Share content directly from Instagram, Pinterest, Tiktok and Youtube via the native share sheet
- Support further platforms/sources via direct URL or image URL saving.
- Categorise content into custom collections and tag posts for semantic retrieval
- Search through collections using keywords and tag-based filters.
- View embedded post previews in-app using oEmbed API where native API access is restricted.
- Maintain persistence across user sessions and real-time synced data via Firebase

Beyond the initial MVP scope, the project also introduced several innovative contributions:

- A hybrid recommendation system blending NLP-based content relevance with fallback recency logic
- A platform-detection parser for share-intent flows, automating post creation
- Embed wrappers with fallback screens, maintaining UX flow even for unsupported platforms.
- A Firebase-Cloudinary architecture enabling scalable and low-cost media access.
- Statistics visualisation, ability to search for other users' collections, bookmarking, and onboarding personalisation for richer UX.

These features extend the scope beyond a basic aggregator and position Collecti as a novel tool for unified content curation.

9.2 Strengths and Weaknesses

9.2.1 Strengths

Extensive consideration was given with regards to creating a user-centric UX, with features such as onboarding flows, clarity of navigation and progressive disclosure of advanced features. Fallback resilience was implemented to support error scenarios, including failed embeds or unsupported platform interactions. Visual hierarchy was maintained, with a developed feedback mechanism of toast notifications, ensuring smooth onboarding for non-technical users. Manual tagging as well as metadata extraction provides flexibility in content organisation, supporting meaningful cross-platform retrieval. The support of different media types such as videos, images or links across platforms is

also worth noting as a large strength. Furthermore, using a Firebase hierarchical schema and Cloudinary for media storage ensures cost-efficiency and scalability of the project.

9.2.2 Weaknesses

API limitations restrict the depth of data retrievable from some platforms, which in turn increases the need for manual tagging by users. This reduces automation and may lead to inconsistencies in how content can be organised. oEmbed solutions for this case offer limited support for metadata extraction and limited control over embed styling. Additionally, the recommendation system is based purely on content similarity and not considerate of user interactions and engagement analytics - it could have much more personalisation depth added to it. Another weakness is that the app doesn't have offline support and caching - it is dependent on a stable internet connection, which could limit usability in scenarios of poor connectivity.

9.3 Comparison to Existing tools

Collecti offers a unique synthesis of flexibility, automation and usability which differentiates it from existing tools (Chapter 2), reflecting practical value and underlying innovation.It acts as a cross-platform meta-layer for content aggregation, overcoming siloed limitations of tools like Pinterest, which limits metadata automation and searching for boards/'collections', or Notion, which demands significant setup. With Collecti, users can fully annotate and tag saved content, while still avoiding cognitive overload. Collecti is also not bound to a single domain (like Paprika). Instead, it is content-agnostic, accommodating all content types and reflecting real-world media consumption and saving behaviour.

Chapter 10: **Conclusion**

10.1 Conclusion

This project addressed a clear gap in digital media organisation: the lack of a seamless, cross-platform content aggregation tool that satisfies all user requirements. This includes: being lightweight, user-friendly and adaptable to diverse digital habits. Through an original combination of cross-platform integration, metadata enrichment, fallback resilience and recommendation logic, Collecti goes beyond a basic aggregator to offer a thoughtful, scalable and extensible solution. Developing this mobile application marked a significant milestone, requiring the navigation of unfamiliar technologies, platform constraints and design trade-offs. Despite the challenges, the project was a successful attempt at building a solution on a small scale, allowing to create a project not just with functionality, but also conceptuality and technical maturity, laying a strong foundation for future innovation in this scope.

10.2 Further work

Despite the achievements, several avenues still exist for enhancing the application further. An AI-driven metadata extraction model could be introduced, which would mitigate reliance on manual tagging through a large language model that parses shared links, extracting relevant metadata and suggesting appropriate collection placement. ML models could also be leveraged, trained on user behaviour or content similarity to suggest relevant tags or auto-completions during saving.Where platform terms permit, future iterations could explore deeper integrations via official APIs for enabling access to richer post metadata or media extraction beyond oEmbed. Another potential idea could be enabling collaborative collection-building and the ability to share collections publicly with other users (essentially support for sharing from the Collecti app).

References

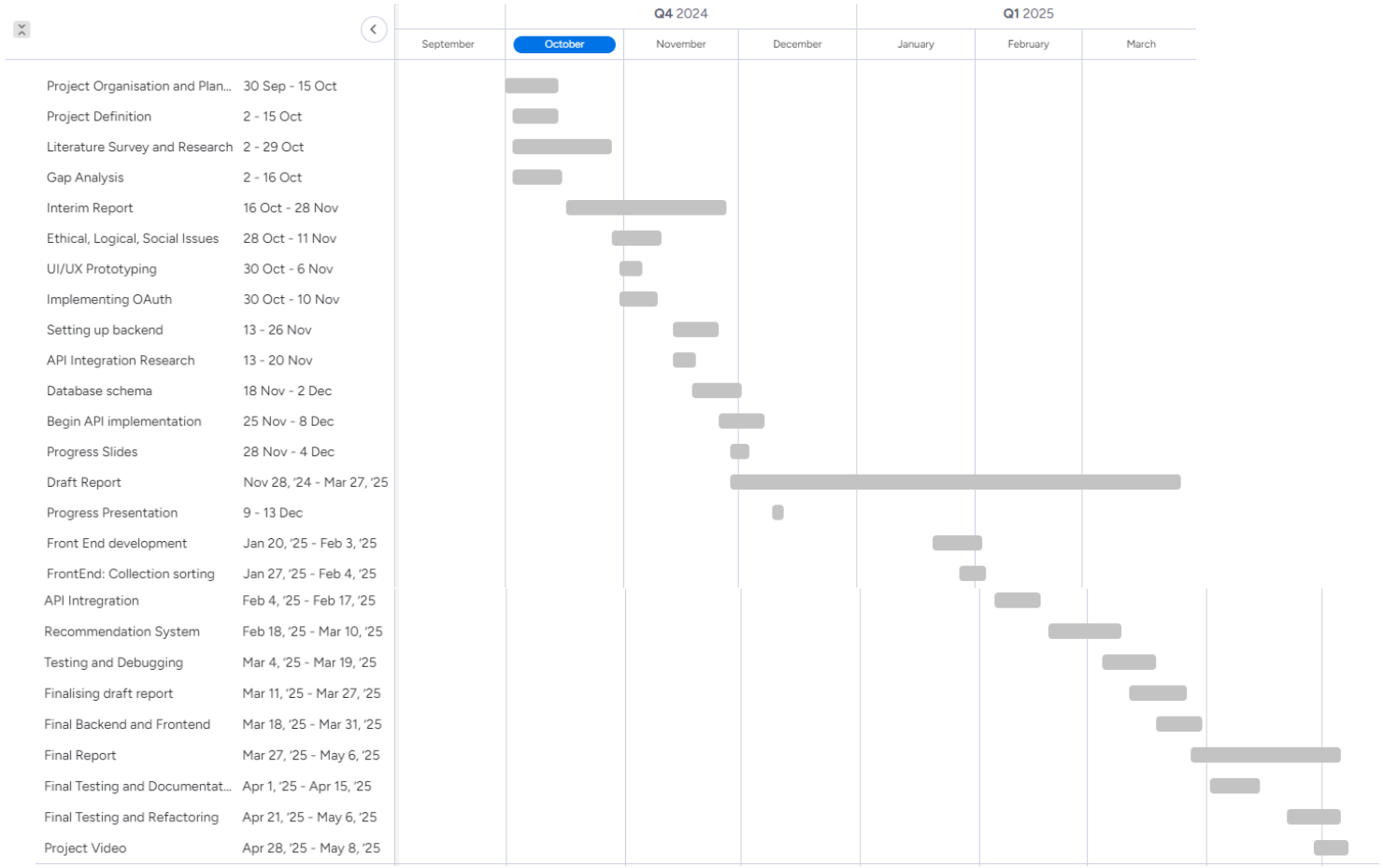
- DataReportal (2024). *Digital 2024: Global Overview Report*. Available at: https://datareportal.com/reports/digital-2024-global-overview-report [Accessed: 18 November 2024]
- Hu, Y., Manikonda, L., & Kambhampati, S. (2014). *What we Instagram: A first analysis of Instagram photo content and user types. Proceedings of the Eighth International AAAI Conference on Weblogs and Social Media.*
- Zarro, M., & Hall, C. (2012). *Exploring social curation as a model for user engagement. Proceedings of the American Society for Information Science and Technology,49(1),1-9.*
- Meta. (2024)**a.** *Instagram Basic Display API Documentation*. Available at: https://developers.facebook.com/docs/instagram-basic-display-api/ [Accessed: 3 October 2024]
- Meta. (2024)**b.** *Instagram oEmbed API Documentation*. Available at: https://developers.facebook.com/docs/instagram-platform/oembed [Accessed: 30 October 2024]
- Meta. (2024)**c.** *Instagram Graph API Documentation*. Available at: https://developers.facebook.com/docs/instagram-platform/instagram-api-with-facebook-login [Accessed: 8 November 2024]
- RapidAPI. (2024). *Instagram Scraper API Documentation.* Available at: https://rapidapi.com/mrngstar/api/instagram-scraper-api3 [Accessed: 8 November 2024]
- Pinterest. (2024)**a.** *Pinterest API Documentation.* Available at: https://developers.pinterest.com/docs/api/v5/introduction/? [Accessed: 30 October 2024]
- Pinterest. (2024)**b** *Pinterest OAuth Tutorial*. Available at: https://pinterest-oauth-tutorial.glitch.me/ [Accessed: 16 November 2024]
- Pinterest. (2024)**c.** *Pinterest API Quickstart*. Available at: https://github.com/pinterest/api-quickstart [Accessed: 11 November 2024]
- AChorein. (2024). *expo-share-intent*. Available at: https://github.com/achorein/expo-share-intent [Accessed: 8 November 2024]
- Expo. (2024). *Expo Go*. Expo. Available at: https://expo.dev/go [Accessed: 5 October 2024]
- React Native. (2024). *Environment Setup*. React Native. Available at: https://reactnative.dev/docs/environment-setup [Accessed: 8 November 2024]
- TikTok. (2024a). *TikTok for Developers*. TikTok. Available at: https://developers.tiktok.com/ [Accessed: 3 October 2024]
- TikTok. (2024b). *Display API Overview*. TikTok. Available at: https://developers.tiktok.com/doc/display-api-overview?enter_method=left_navigation [Accessed: 3 October 2024]

- TikTok. (2024c). *Embed Player Documentation*. TikTok. Available at: https://developers.tiktok.com/doc/embed-player?enter_method=left_navigation [Accessed: 30 October 2024]
- TikTok. (2024d). *Embed Videos Documentation*. TikTok. Available at: https://developers.tiktok.com/doc/embed-videos?enter_method=left_navigation [Accessed: 30 October 2024]
- Teather, D. (2024). *TikTok API Unofficial Documentation*. Available at: https://davidteather.github.io/TikTok-Api/ [Accessed: 3 October 2024]
- Google. (2024a). YouTube Data API v3. Available at: https://developers.google.com/youtube/v3/docs [Accessed 30 October 2024]
- Google. (2024b). YouTube IFrame Player API. Available at: https://developers.google.com/youtube/iframe_api_reference [Accessed 25 November 2024]
- Pentina, I., & Tarafdar, M. (2014). *From Information to Knowing: Exploring the Role of Social Media in Contemporary News Consumption*. *International Journal of Information Management*, 34(3), 300-310.
- Bates, M. J. (1989). *The Design of Browsing and Berrypicking Techniques for the Online Search Interface*. *Online Review*, 13(5), 407-424.
- Furnas, G. W., Landauer, T. K., Gomez, L. M., & Dumais, S. T. (1987). The vocabulary problem in human-system communication. *ACM Transactions on Computer-Human Interaction*, 3(2), 185-223.
- Wikipedia. (2024). Ontology (Information Science). Available at: https://en.wikipedia.org/wiki/Ontology_(information_science) [Accessed 8 November 2024]
- Heath, T. and Bizer, C., 2011. *Linked Data: Evolving the Web into a Global Data Space*. In A. Passant, ed. *Foundations for the Web of Information and Services*. Springer, pp. 5-2.
- Hardt, D., 2012. *The OAuth 2.0 Authorization Framework*. RFC 6749. Internet Engineering Task Force (IETF). Available at: https://datatracker.ietf.org/doc/html/rfc6749 [Accessed 25 November 2024]
- Descope, 2024. *Descope Documentation*. Available at: https://descope.com [Accessed 18 November 2024]
- Google, 2024. *Firebase Authentication.* Available at: https://firebase.google.com/docs/auth [Accessed 25 November 2024]
- OAuth.net, 2024. *OAuth 2.0 Authorization Framework*. Available at: https://oauth.net/2/ [Accessed 25 November 2024]
- Retail & Hospitality Information Sharing and Analysis Center (RH-ISAC), 2024. *Application Security Compliance Standards*. Available at: https://rhisac.org/application-security/application-security-compliance-standards/ [Accessed 25 November 2024]
- Krug, S. (2000). Don't Make Me Think: A Common Sense Approach to Web Usability. 1st ed. Indianapolis: New Riders.

- Shneiderman, B. (2000). Universal Usability. Communications of the ACM, 43(5), pp. 84–91.
- Norman, D.A. (1988). The Design of Everyday Things. Revised and Expanded Edition. New York: Basic Books.
- W3C (World Wide Web Consortium), 2018. *Web Content Accessibility Guidelines (WCAG) 2.1.* [online] Available at: https://www.w3.org/TR/WCAG21/ [Accessed 27 November 2024].
- Lidwell, W., Holden, K., & Butler, J. (2010). *Universal Principles of Design* (2nd ed.). Rockport Publishers.
- Csikszentmihalyi, M. (1990). *Flow: The Psychology of Optimal Experience*. Harper & Row.
- Heidegger, M. (1927). *Being and Time*.
- Lops, P., De Gemmis, M., & Semeraro, G. (2011). *Content-based Recommender Systems: State of the Art and Trends*. In Recommender Systems Handbook. Springer.

Appendix A

Gantt Chart and Project Plan



Appendix B

Key Risk Assessment points

RISK 1

RISK: FAILURE TO ACCESS REQUIRED APIS

LIKELIHOOD: Medium

SEVERITY: High

IMPACT:
Unable to gather content from social media platforms

PREVENTATIVE/MITIGATING ACTIONS:

- Ensure API access is setup and **granted early** on and monitor for limitations
- Maintain **backup options** for data retrieval such as third-party APIs or embed API workarounds

RISK 2

RISK: USER INTERFACE IS NOT ACCESSIBLE FOR DIVERSE USER GROUPS

LIKELIHOOD: Low

SEVERITY: Medium

IMPACT:
reduced usability, and final product may not meet user preferenfces/expectations

PREVENTATIVE/MITIGATING ACTIONS:

- During the design process adhere to a**ccessibility standards** (WCAG 2.2) and refer to expected system requirements

RISK 3

RISK: FAILURE TO TEST AND DEBUG APP ADEQUATELY

LIKELIHOOD: Low

SEVERITY: Medium

IMPACT:

Bugs that prevent functionaility (e.g in API or database)

PREVENTATIVE/MITIGATING ACTIONS:

- **Review** code regularly and **debug**
- **Unit test** every new code implementation
- Perform function AND usability testing with potential users

RISK 4

RISK: POOR TIME MANAGEMENT

LIKELIHOOD: Medium

SEVERITY: High

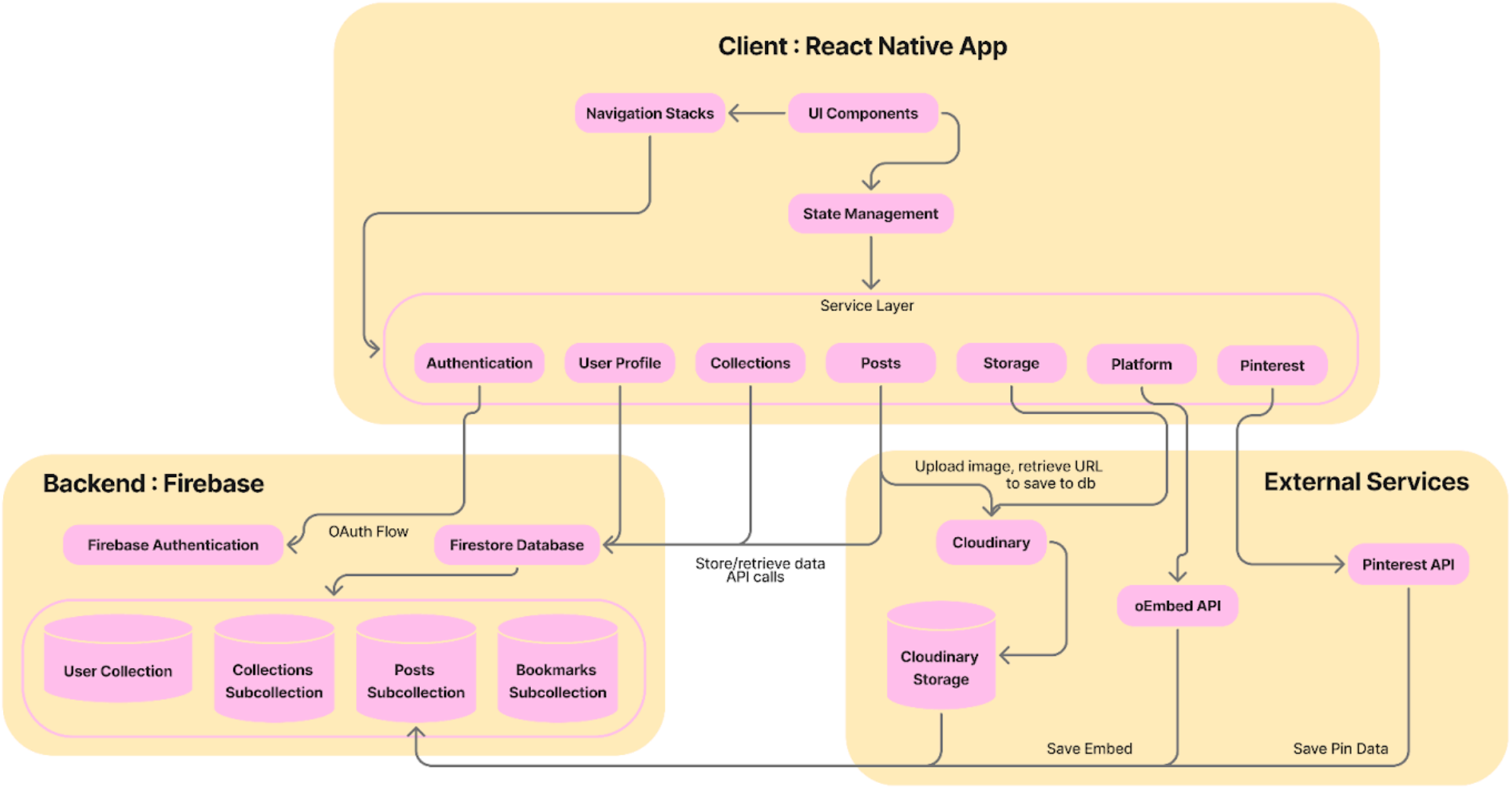
IMPACT:

Project fall behind schedule, impact on final deliverables, not able to reach all proclaimed milestones

PREVENTATIVE/MITIGATING ACTIONS:

- Following **timeplan**, allocating time for each phase
- Setting clear **milestones**, earlier than deadlines with a fallback/safety net period before actual deadlines

Appendix C


Client : React Native App
Navigation Stacks
UI Components
State Management
Service Layer
Authentication
User Profile
Collections
Posts
Storage
Platform
Pinterest
Backend : Firebase
Firebase Authentication
OAuth Flow
Firestore Database
User Collection
Collections Subcollection
Posts Subcollection
Bookmarks Subcollection
Store/retrieve data API calls
Upload image, retrieve URL to save to db
Cloudinary
Cloudinary Storage
oEmbed API
External Services
Pinterest API
Save Embed
Save Pin Data

Appendix D

User Testing Google Form

Section 1: Basic Info

How often do you save/favourite content from social media? *

- ○ Daily
- ◉ Weekly
- ○ Occasionally
- ○ Never

If you do save content, which platforms do you usally save content from? (Select all that apply) *

- ☐ Instagram
- ☐ Tiktok
- ☐ Youtube
- ☐ Pinterest
- ☐ Twitter
- ☐ Facebook
- ☐ Other: _____

Section 2: Task completion

Which of the following tasks did you attempt? *

- [ ] Create an account
- [ ] Complete the onboarding process.
- [ ] Create a new collection.
- [ ] From any supported platform (Instagram, TikTok, YouTube, Pinterest), find content you like and share it to the Collecti app
- [ ] Add a description, some tags, and assign it to a collection (or leave it in the default "unsorted" collection)
- [ ] View the post inside the collection
- [ ] Edit the post or change the tags
- [ ] Edit the collection name or update the description
- [ ] Explore the app navigation — switch between tabs and see what's available
- [ ] Add another post, either from your device's gallery or by pasting an image URL
- [ ] Go to a collection and select a post and move it to a different collection.
- [ ] Delete a post from a collection
- [ ] Delete a collection
- [ ] Search for a post using keywords or tags
- [ ] Search for a collection
- [ ] Try sharing a post from a less-supported platform like Twitter
- [ ] Check if links open properly from saved posts
- [ ] Log out and log back in

Out of the ones you attempted, which ones were you able to **complete successfully** (as expected)? *

- [ ] Create an account
- [ ] Complete the onboarding process.
- [ ] Create a new collection.
- [ ] From any supported platform (Instagram, TikTok, YouTube, Pinterest), find content you like and share it to the Collecti app
- [ ] Add a description, some tags, and assign it to a collection (or leave it in the default "unsorted" collection)
- [ ] View the post inside the collection
- [ ] Edit the post or change the tags
- [ ] Edit the collection name or update the description
- [ ] Explore the app navigation — switch between tabs and see what's available
- [ ] Add another post, either from your device's gallery or by pasting an image URL
- [ ] Go to a collection and select a post and move it to a different collection.
- [ ] Delete a post from a collection
- [ ] Delete a collection
- [ ] Search for a post using keywords or tags
- [ ] Search for a collection
- [ ] Try sharing a post from a less-supported platform like Twitter
- [ ] Check if links open properly from saved posts
- [ ] Log out and log back in

Section 3: Usability & Feature Feedback

How easy was it to use the app/complete the tasks overall? *

1 2 3 4 5

☆ ☆ ☆ ☆ ☆

How visually appealling did you find the app? *

1 2 3 4 5

☆ ☆ ☆ ☆ ☆

Did you understand how to share content from social media into Collecti? *

- ◯ Yes, it was straightforward
- ◯ I figured it out eventually
- ◯ No, I was confused
- ◯ Didn't try this

Did the collections feature help you organise your saved content effectively? *

- ◯ Yes
- ◯ Kind of
- ◯ Not really
- ◯ No

Were you able to find your saved posts again easily after adding them? *

- ◯ Yes
- ◯ Somewhat
- ◯ No
- ◯ Didn't try

Section 4: General Impressions

What was your favourite part of the app?

Your answer

Was anything confusing or unclear or not working as you expected?

Your answer

Was anything confusing or unclear or not working as you expected?

Your answer

Would you use an app like Collecti in real life? Why or why not?

Your answer

How would you rate this app overall? *

1 2 3 4 5

☆ ☆ ☆ ☆ ☆

How useful do you think this app is? *

1 2 3 4 5

☆ ☆ ☆ ☆ ☆

Anything else youd like to share?

Your answer

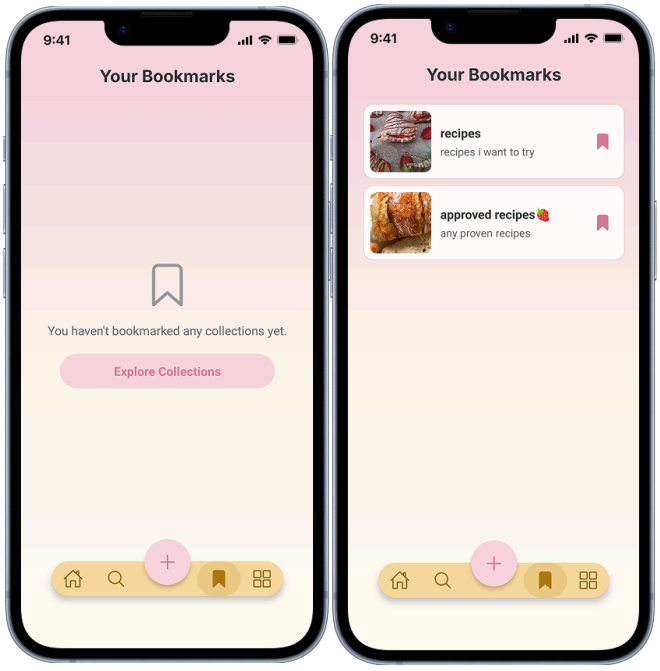
Submit

Page 1 of 1

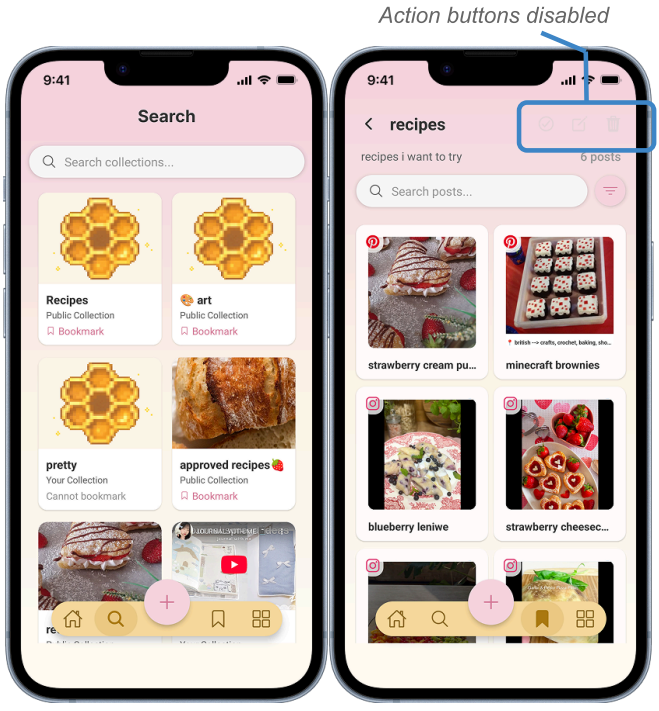
Clear form

Appendix E

Additional Key Screens

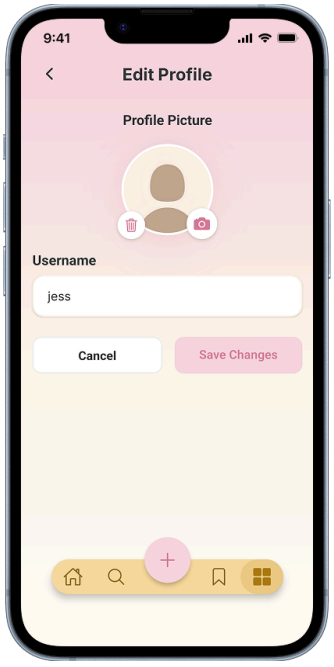


Bookmarks Screen (empty vs. saved)

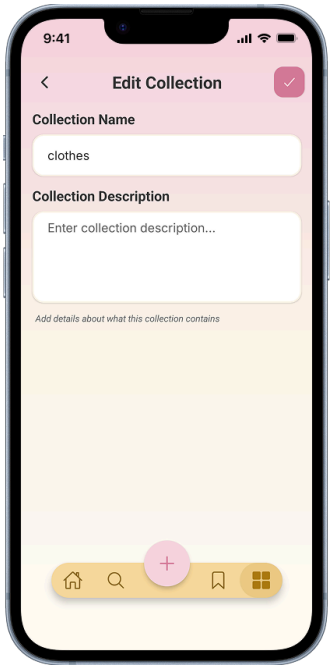


Search Screen

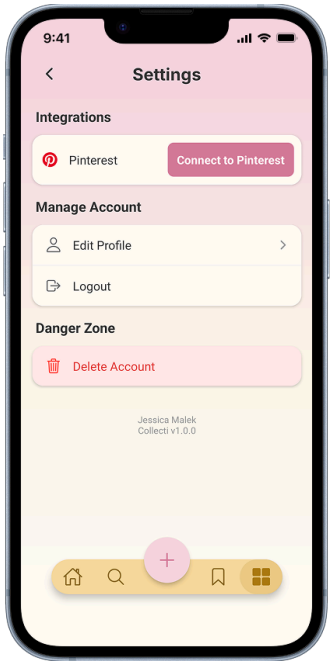
Another User's Collection



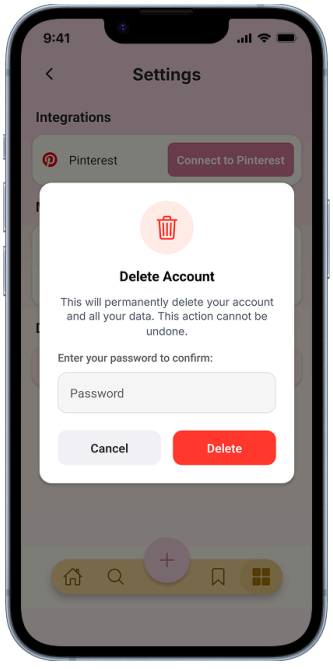
Edit Profile Screen



Edit Collection/Post Screen



Settings Screen



Delete Account Screen