A Web-Based Social Media Application – Mask: Share Freely, Trust Wisely

By
Md. Tanvir Tahmid
213-15-4425

FINAL YEAR DESIGN PROJECT REPORT
This Report is Presented in Partial Fulfillment of the Requirements for the **Degree of Bachelor of Science in Computer Science and Engineering**

Supervised by
Ms. Tapasy Rabeya
Lecturer (Senior Scale)
Department of Computer Science and Engineering
Daffodil International University

Co-Supervised by
Mr. Md. Abbas Ali Khan
Assistant Professor
Department of Computer Science and Engineering
Daffodil International University

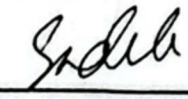
DAFFODIL INTERNATIONAL UNIVERSITY
Dhaka, Bangladesh
September 17, 2025

APPROVAL

This Project titled **"A Web-Based Social Media Application – Mask: Share Freely, Trust Wisely"**, submitted by **Md. Tanvir Tahmid**, ID No: **213-15-4425** to the Department of Computer Science and Engineering, Daffodil International University has been accepted as satisfactory for the partial fulfillment of the requirements for the degree of B.Sc. in Computer Science and Engineering and approved as to its style and contents. The presentation has been held on **16 September, 2025**.

BOARD OF EXAMINERS

Dr. Sheak Rashed Haider Noori — **Chairman**
Professor and Head
Department of CSE, FSIT
Daffodil International University



Mr. Md. Sadekur Rahman — **Internal Examiner**
Assistant Professor
Department of CSE, FSIT
Daffodil International University

Mr. Saiful Islam — **Internal Examiner**
Assistant Professor
Department of CSE, FSIT
Daffodil International University

Dr. Md. Arshad Ali — **External Examiner**
Professor
Department of CSE
Hajee Mohammad Danesh Science & Technology University

DECLARATION

We hereby declare that this project has been done by us under the supervision of Ms. Tapasy Rabeya, Lecturer (Senior Scale) Department of Computer Science and Engineering, Daffodil International University. We also declare that neither this project nor any part of this project has been submitted elsewhere for the award of any degree or diploma.

Supervised by:

Ms. Tapasy Rabeya
Lecturer (Senior Scale)
Department of Computer Science and Engineering
Daffodil International University

Co-Supervised by:

15.9.25

Mr. Md. Abbas Ali Khan
Assistant Professor
Department of Computer Science and Engineering
Daffodil International University

Submitted by:

15-09-2025

Md. Tanvir Tahmid
Student ID: 213-15-4425
Department of Computer Science and Engineering
Daffodil International University

ACKNOWLEDGEMENTS

This work would not have been possible without the support and contributions of many individuals over the past two semesters. We are deeply grateful to everyone who has assisted us in one way or another.

First, we express our heartfelt thanks and gratefulness to the Almighty for His divine blessing making it possible for us to complete the Final Year Design Project(FYDP) successfully.

We are grateful and wish to express our profound indebtedness to Ms. Tapasy Rabeya, Lecturer (Senior Scale), Department of Computer Science and Engineering, Daffodil International University, Dhaka, Bangladesh. Deep knowledge and keen interest of our supervisor in the field of Web Application to carry out this project. His endless patience, scholarly guidance, continual encouragement, constant and energetic supervision, constructive criticism, valuable advice, reading many inferior drafts, and correcting them at all stages have made it possible to complete this project.

We would like to express our heartfelt gratitude to the Head of the Department of Computer Science and Engineering, for his kind help in finishing our project and also to other faculty members and the staff of the Department of Computer Science and Engineering, Daffodil International University.

We would like to thank our entire course-mates at Daffodil International University, who took part in this discussion while completing the coursework.

Finally, We are deeply grateful to our parents, whose constant support and patience made it possible for us to see this project through.

ABSTRACT

Mask App is a full-stack social networking platform built to test how modern moderation tools and credibility signals can fit naturally into a familiar social feed. It offers all the expected features-posting, commenting, reacting, sharing, bookmarking, and creating groups or pages with scoped visibility-while adding real-time notifications and private messaging secured with AES-GCM encryption. A server-side link-unfurl pipeline generates rich previews for shared URLs, while an admin console supports policy enforcement and content removal across user, page, and group contexts. A distinguishing element of Mask App is its built-in fact-check and trust scoring workflow. Text posts may be evaluated by an external model (Google Gemini) through a backend API. Checks are triggered by engagement thresholds or an optional author tag and are guarded by rate budgets to control cost and load. The model returns a structured verdict with an explanation and confidence, which is surfaced as a compact "context pill" on the post. Separately, the system aggregates historical outcomes into an author-level trust score and tier. This score drives a visible Trust Badge and enables ranking adjustments in discovery views, offering users lightweight, interpretable signals without hiding content outright. Architecturally, the frontend is a React application, and the backend is a Node/Express API backed by MongoDB, with optional Cloudinary storage. The project is deployable on common student-friendly platforms (e.g., Vercel for the frontend and Render for the backend) using environment-driven configuration. Overall, Mask App demonstrates a practical approach to credibility features in social media while preserving usability. Future work includes broader multilingual checks, richer analytics for admins, and expanded privacy controls for messaging.

Table of Contents

APPROVAL ii
DECLARATION iii
ACKNOWLEDGEMENTS iv
ABSTRACT v
Table of Contents vi
List of Figures viii
List of Tables ix
Introduction 1
1.1 Introduction 1
1.2 Motivation 1
1.3 Objectives 2
1.4 Methodology 2
1.5 Project Outcome 3
1.6 Organization of the Report 3
Background 5
2.1 Introduction 5
2.2 Literature Review 5
2.2.1 Similar Applications 11
2.3 Gap Analysis 13
2.4 Summary 14
Research Methodology 16
3.1 Requirement Analysis & Design Specification 16
3.1.1 Overview 16
3.1.2 System Design 17
3.1.3 Component & Service Architecture 19
3.1.4 Functional and Nonfunctional Requirements 21
3.1.5 Data Flow Diagram 25
3.1.6 Entity-Relationship (ER) Diagram 27
3.1.7 UI Design 30
3.2 Detailed Methodology and Design 43
3.3 Project Plan 45

3.4 Task Allocation 47
3.5 Summary 47
Implementation and Results 49
4.1 Environment Setup 49
4.2 Testing and Evaluation 50
4.3 Results and Discussion 53
4.4 Summary 54
Engineering Standards and Design Challenges 55
5.1 Compliance with the Standards 55
5.2 Impact on Society, Environment and Sustainability 56
5.2.1 Impact on Life 57
5.2.2 Impact on Society & Environment 57
5.2.3 Ethical Aspects 57
5.2.4 Sustainability Plan 57
5.3 Project Management and Financial Analysis 58
5.4 Complex Engineering Problem 58
5.4.1 Complex Problem Solving 59
5.5 Summary 63
Conclusion 64
6.1 Summary 64
6.2 Limitation 64
6.3 Future Work 65
References 67

List of Figures

3.1 Mask Use Case Diagram…….. 17

3.2 Mask App - Component & Service Architecture Diagram…….. 19

3.3 Data Flow Post Creation Diagram……... 24

3.4 Data Flow - Direct Messages Stored Diagram…….. 25

3.5 ER diagram of User Domain…… 26

3.6 ER diagram of Admin Domain…….. 26

3.7 MaskDB Atlas Diagram…….... 27

3.8 Sign-up page…….. 30

3.9 Log in page…… 30

3.10 Homepage of Mask…….... 31

3.11 Auto-expire posts (TTL)…….. 33

3.12 Inline fact-check pill…….. 35

3.13 Limitations of the inline fact-check pill……... 36

3.14 Author trust score visible…….. 38

3.15 Wellbeing nudges…….. 40

3.16 Forced Log Out…….. 41

3.17 Daily Motivation Quote…….. 42

List of Tables

2.1 Summary of Literature Reviewed..... 6

2.3 Gap Analysis of Existing vs Proposed System..... 12

3.1 Task Allocation..... 46

4.1 Test Cases and Results..... 49

5.1 Project Cost and Financial Analysis..... 57

5.2 Mapping with complex problem solving..... 58

5.3 Mapping with Knowledge Profile..... 59

5.4 Mapping with complex engineering activities..... 61

Chapter 1

Introduction

This chapter presents an overview of the Mask App project. It explains why the project was conceived, defines its objectives, summarizes the approach used to build it, describes the expected outcomes, and outlines how the remainder of the report is organized.

1.1 Introduction

Social media connects millions of people and makes it effortless to share photos, videos, daily activities, and community work. Yet the same design patterns that make platforms engaging can also encourage unbounded scrolling and amplify low-value or misleading content. Feeds rarely "end," and outdated items can resurface long after they are useful. Mask App explores a different balance: it keeps the familiar social experience but adds two guardrails-time-bounded visibility and lightweight credibility signals. By letting posts expire after a sensible period and by showing concise context on claims, Mask aims to reduce wasted time while helping users make quicker, more informed judgments. The approach is deliberately gentle: nothing is hidden by default, but stale items are removed, and factual claims can be annotated with clear, interpretable signal.

1.2 Motivation

This project began with a personal observation: a large share of screen time was spent in feeds with no natural stopping point. Old posts continued to appear, and there was always "one more" item to check. That felt inefficient and, at times, distracting. I wondered what would happen if the feed were intentionally finite, with content that expires after a reasonable window. The idea matured as I noticed a second issue: even when time is limited, it is hard to judge the reliability of claims inside a fast-moving feed. That led to combining two ideas-retention rules that remove stale posts to prevent endless loops, and credibility signal that help users understand the status of a claim without leaving the app. Together, these interventions favor recency and clarity over volume, supporting healthier usage without taking away user choice or pseudonymity.

1.3 Objectives

The project's objectives are grouped into user experience, credibility, privacy, and practicality:

Time efficiency: apply post retention (time-to-live) so stale items are removed and the feed naturally resets, reducing endless scrolling loops.

Credibility support: evaluate text claims via a server-side fact-check pipeline and present a compact verdict pill (with explanation and confidence).

Author trust score: aggregate historical outcomes into a simple Trust Badge and use it to gently adjust discovery ranking, keeping signals transparent.

Pseudonymity with accountability: support pseudonymous profiles to lower social pressure while preserving moderation and auditability.

Core social features: posts, reactions, comments, shares, bookmarks, groups/pages with scoped visibility, encrypted direct messages, notifications, search, and rich link previews.

Operational safeguards: rate budgets and spacing for external model calls, one-time checks per post, and predictable costs.

Practical deployment: run reliably on student-friendly platforms (e.g., Vercel for frontend, Render for backend) with environment-driven configuration.

Maintainability and extensibility: keep the architecture clear (React + Node/Express + MongoDB), isolate modules (fact-check, link-unfurl, messaging), and document environment variables for future contributors.

1.4 Methodology

Mask App is built on the MERN stack-MongoDB, Express.js, React, and Node.js-chosen for its unified JavaScript workflow, rapid iteration, and suitability for real-time, feed-centric interfaces. The architecture keeps concerns modular: the React client handles stateful UI and routing; the Node/Express API exposes authenticated endpoints; MongoDB stores documents for posts, users, groups/pages, messages, notifications, and fact-check results. Supporting services include a link-unfurl module for rich previews, an AES-GCM messaging layer keyed via environment variables, and a rate-limited fact-check pipeline that calls an external model and persists structured verdicts. Environment-driven configuration (e.g., API origins, cryptographic keys, Cloudinary, model budgets) enables the same codebase to run

locally and on common student-friendly hosts. Project requirements were formalized in a Software Requirements Specification (SRS), which served as the blueprint for architecture, performance targets (latency, throughput), and moderation rules (post retention/TTL, admin scopes).

This report adopts a formal analytic approach. It begins with a focused review of literature and comparable systems to establish academic and market context around time-bounded feeds, credibility signals, and private messaging. It then deconstructs the system design as specified in the SRS: data flows for post creation, link previewing, fact-check triggering and storage, trust-score aggregation, and decryption paths for direct messages. The implementation plan and expected results are presented alongside verification methods (functional tests, scenario checks for retention, authorization, and rate budgets). The discussion evaluates compliance with applicable engineering standards (JWT, HTTPS, CORS), ethical considerations of automated labeling, and potential societal impact in terms of time well-spent and reduced confusion. Finally, limitations and avenues for future work are outlined, connecting the technical description to a broader academic and practical interpretation of the project's significance.

1.5 Project Outcome

The outcome is a working prototype of a social platform that combines time-bounded feeds with credibility indicators. Users create posts, react, comment, share, bookmark, and participate in groups or pages with appropriate visibility. URLs render rich previews; notifications surface relevant activity. When a claim is evaluated, a context pill appears with a verdict and confidence, and authors earn a Trust Badge that reflects longer-term behavior. Admin tools support content removal by owners, global admins, or page/group admins. Expected benefits include: (i) lower time spent on stale items, (ii) clearer interpretation of claims without heavy moderation, and (iii) a clean deployment path useful for student projects and small teams. The prototype also creates a foundation for future study-such as multilingual checks, better explanations for verdicts, richer analytics for admins, and expanded privacy controls for direct messages.

1.6 Organization of the Report

The remainder of this document is organized to guide readers from context to implementation and results. Chapter 2 reviews related work on social feeds, moderation approaches, fact-checking methods, and private messaging, and it identifies the gap Mask aims to address. Chapter 3 details the methodology and system design, including functional and non-functional requirements, component and data-flow diagrams, and the fact-check/trust workflow. There are also UI and other

parts that help visualize the application. Chapter 4 explains the implementation, development environment, and testing results, with tables that summarize scenarios and outcomes. Chapter 5 discusses applicable standards (e.g., JWT, HTTPS, CORS), design challenges, ethical considerations around labeling, and sustainability constraints such as rate budgets and cost. Chapter 6 concludes with a summary of contributions, current limitations, and concrete directions for future work. Appendices provide supplementary figures, tables, and configuration references for deployment

Chapter 2

Background

This chapter situates Mask App within current practice and scholarship. It explains why a credibility-aware, time-bounded feed is relevant, summarizes the literature that informed the design, compares popular platforms, and makes the gaps explicit.

2.1 Introduction

In Bangladesh and elsewhere, social media is the default public square for connection, activism, and commerce-driven by affordable smartphones and a large youth population. Alongside these benefits, three issues repeatedly surface: (1) feeds designed for infinite engagement that encourage unbounded scrolling; (2) the rapid spread of misleading or outdated claims; and (3) limited, transparent signals that help ordinary users assess credibility without leaving the app. Mask App is framed as a pragmatic response. It preserves a familiar experience but adds time-bounded visibility (post TTL/retention) and lightweight credibility signal (a fact-check verdict pill on posts and an author-level Trust Badge), while allowing pseudonymous participation and AES-GCM-secured private messaging. The aim is not heavy censorship; it is to make the feed finite and the signals clearer.

2.2 Literature Review

Recent scholarship paints a consistent picture of social platforms: they excel at connection and discovery but often optimize for engagement length rather than user value. Studies on attention and "infinite scroll" document how feed mechanics and intermittent rewards keep people cycling through older and only weakly relevant posts. That evidence supports Mask's decision to introduce retention/expiry so feeds naturally reset and stale items stop resurfacing. A second strand examines misinformation and verification. Reviews of fact-checking and crisis-era platform use show that blunt removals can be contentious and slow, while concise, on-post context helps ordinary readers make faster judgments. Mask therefore adds a post-level verdict pill and an author-history Trust Badge rather than hiding content. Other work on governance emphasizes scoped authority and transparent rules, which we implement as owner OR global admin OR page/group admin deletion, plus logs. Finally, privacy and safety work argues for secure messaging even in student projects, leading to AES-GCM-protected direct messages and environment-based key management. Together these streams informed a design that aims to reduce unproductive time while adding interpretable credibility signals without removing user choice or pseudonymity.

Table 2.1: Summary of Literature Reviewed.

Author(s)	Year	Title	Methodology	Key Findings
V. Bhatia, T. Jain, H. Choudhary, N. Chawla [1]	2025	Scrolling in Silence: Is Social Media Bridging Connections or Fueling Isolation?	Narrative review	Explores both the positive and negative effects of social media use, emphasizing how endless scrolling can lead to isolation without mindful design.
N. D. Estolatan-Lama, B. E. Isolana, O. R. Castañares Jr. [2]	2025	Social Media: Misinformation and Trustability	Descriptive review	Highlights how misinformation spreads quickly online and calls for stronger media literacy and trust-building practices.
M. Rodríguez-Ibánez, A. Casánez-Ventura, F. Castejón-Mateos, P.-M. Cuenca-Jiménez [3]	2023	A review on sentiment analysis from social media platforms	Survey & bibliometric review	Maps out sentiment analysis methods and datasets, showing how cross-domain applications are increasingly shaping social media research.
H. Taherdoost [4]	2023	Enhancing social media platforms with machine learning algorithms and neural networks	Algorithmic survey	Summarizes ML/AI techniques for moderation, recommendation, and personalization, noting both potential and challenges.

D. V. Gunasekeran *et al.* [5]	2022	The impact and applications of social media platforms for public health during COVID-19	Systematic literature review	Shows how social media supported communication and tracking during the pandemic, but also fueled misinformation and risks.
T. Aichner, M. Grünfelder, O. Maurer, D. Jegeni [6]	2021	Twenty-Five Years of Social Media: A Review of Applications and Definitions	Structured literature review	Provides a taxonomy and evolving definitions of social media, useful for viewing newer platforms like Mask App.
M. D. Szeto, A. Mamo, A. Afrin, M. Militello, C. Barber [7]	2021	Social media in dermatology and an overview of popular platforms	Review paper	Demonstrates social media's role in education and outreach, while also warning of misinformation in health contexts.
I. Owuor, H. H. Hochmair [8]	2020	An overview of social media apps and their potential role in geospatial research	Review paper	Describes how social platforms provide geospatial data for research, but also notes privacy and accuracy concerns.
M. Brough, I. Literat, A. Ikin [9]	2020	"Good Social Media?": Youth Perspectives on Ethical and Equitable Design	Qualitative study	Captures young users' voices, highlighting the demand for safer and fairer design in social networks.

C. Gerlitz, A. Helmond, F. N. van der Vlist, E. Weltevrede [10]	2019	Regramming the Platform: Infrastructural Relations	Empirical platform analysis	Examines how apps and third parties reshape social media infrastructures beyond platform control.
A. Helmond, F. N. van der Vlist [11]	2019	Social Media and Platform Historiography	Conceptual paper	Discusses methods for studying platform history, noting challenges in data access and archival work.
M. Naeem [12]	2019	Cross-platform apps as tools for knowledge sharing	Qualitative study	Shows how university students use social apps for cross-platform knowledge sharing, stressing adoption and collaboration.
C. Montag, B. Lachmann, M. Herrlich, K. Zweig [13]	2019	Addictive Features of Social Media/Messenger Platforms	Conceptual review	Identifies design elements that drive addictive use, comparing them with freemium games and economic theories.
A. A. Alalwan, N. P. Rana, Y. K. Dwivedi, R. Algharabat [14]	2017	Social media in marketing: A review and analysis	Systematic literature review	Analyzes determinants and effects of social media in marketing, noting opportunities and risks for businesses.

S. Picazo-Vela, I. Gutiérrez-Martínez, L. F. Luna-Reyes [15]	2012	Risks, benefits, and strategic alternatives of social media in public sector	Conceptual framework	Explains risks and benefits of government adoption of social apps, recommending cautious but strategic integration.
S. Vosoughi, D. Roy, S. Aral [16]	2018	The spread of true and false news online	Large-scale Twitter analysis	Finds that false news spreads faster and wider than true news, raising concerns about misinformation dynamics.
D. M. J. Lazer *et al.* [17]	2018	The science of fake news	Interdisciplinary review	Argues that tackling fake news requires both technical solutions and broader social/policy interventions.
H. Allcott, M. Gentzkow [18]	2017	Social media and fake news in the 2016 election	Survey and data analysis	Demonstrates how fake news gained significant attention during the U.S. election, shaping debate around platform responsibility.
K. Shu, A. Sliva, S. Wang, J. Tang, H. Liu [19]	2017	Fake news detection on social media: A data mining perspective	Survey of methods	Reviews detection strategies, emphasizing that combining linguistic, network, and user

				features works best.
C. Wardle, H. Derakhshan [20]	2017	Information Disorder: Toward an Interdisciplinary Framework	Policy report	Frames misinformation as "information disorder," highlighting the role of education and governance beyond algorithms.
M. J. Dworkin (NIST) [21]	2007	Recommendation for Block Cipher Modes of Operation: GCM and GMAC	Cryptography standard	Defines AES-GCM as a secure, efficient encryption method, now standard in modern communication systems.
W3C [22]	2021	WebRTC 1.0: Real-Time Communication Between Browsers	Web standard	Provides the foundation for real-time media features like video calls and live streams in browsers.
I. Fette, A. Melnikov [23]	2011	The WebSocket Protocol (RFC 6455)	Internet standard	Establishes WebSockets as a way to keep open, two-way connections for live feeds and instant updates.
M. Cha, H. Haddadi, F. Benevenuto, K. Gummadi [24]	2010	Measuring user influence in Twitter: The million follower fallacy	Empirical study	Shows that follower counts alone do not reflect influence; engagement

				signals like retweets are stronger indicators.
D. Boyd, N. Ellison [25]	2007	Social network sites: Definition, history, and scholarship	Scholarly review	One of the first academic definitions of social networking sites, explaining how they shape identity and communities.

2.2.1 Similar Applications

The design of Mask App was shaped by looking closely at the strengths and shortcomings of today's most popular social platforms. Studying these systems helped identify what works well and, just as importantly, what is missing. Mask deliberately takes a different path by introducing a finite feed, adding clear on-post context for claims, showing an author-level Trust Badge, and supporting pseudonymous participation features that set it apart from the endless, engagement-driven models of mainstream networks.

Facebook - A massive, general-purpose network with mature Groups/Pages, events, marketplace, and live streaming. Its ranking is excellent for engagement and discovery, but ordinary posts do not expire by design, and inline truthfulness cues are limited; credibility signals are scattered across policy notices rather than standardized, compact labels.

X (Twitter) - Real-time microblogging with hashtags, trending topics, and broadcast-style accounts. Some posts feature community notes, but this is coverage-limited, and there is no cross-post author trust tier visible to readers. Posts persist, so the feed naturally drifts without a time boundary.

Instagram - Visual-first sharing with reels, stories, and creator tools. **Stories** are ephemeral, yet the core feed remains open-ended; credibility labeling is minimal and not designed for textual claims in captions. Discovery is strong, which can lengthen viewing sessions.

TikTok - Short-form video with powerful recommendation and creation tools (filters, edits, effects). The system maximizes watch time; videos do not expire,

and there are no author trust tiers or standardized on-post fact labels for general claims.

YouTube - Long-form video and livestreaming with monetization and content-ID for copyright. While policy banners appear in some contexts, there is no consistent post-level verdict for claims and no author trust summary for viewers; content longevity is a core feature.

Reddit - Topic-centric communities with volunteer moderation and up/down voting. Excellent for deep discussion and niche interests, but threads and posts don't expire by default, and there is no global author credibility badge across communities.

Snapchat - Close-circle social and ephemeral stories with strong AR lens effects. Public, claim-oriented feeds and credibility labels are not its focus; it does not aim to manage attention for longer discussions.

LinkedIn - Professional network with real-identity norms, job tools, and industry communities. Emphasis on workplace relevance but no feed TTL or compact claim evaluation signals; identity verification exists but is not a truthfulness indicator.

Pinterest - Visual discovery and curation system optimized for topics and projects; excellent at saving/organizing, not at real-time claim evaluation or feed finality.

Telegram (Channels) - Broadcast feeds with large audiences; good for rapid distribution and pseudo-public posts, yet channel content persists and credibility signal depend entirely on admins or external links.

Where Mask positions itself: a finite, recency-aware feed; on-post verdict pill for claims; an author Trust Badge derived from history; pseudonymous accounts with AES-GCM DMs-keeping familiar social features while injecting gentle guidance rather than heavy takedowns.

2.3 Gap Analysis

Here put summarize the gaps you have found/observed from the related work study, where you intend to contribute.

Table 2.3: Gap Analysis of Existing vs Proposed System

Feature / Criterion	Facebook	X (Twitter)	Instagram	TikTok	YouTube	Snapchat	LinkedIn	Mask App
User profiles	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Comments & reactions, Notifications	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Direct messaging	Yes (Messenger)	Yes	Yes	Yes	No (discontinued)	Yes (core)	Yes	Yes (AES-GCM)
Followers/ friends graph	Yes	Yes	Yes	Yes	Yes (subs)	Yes	Yes	Yes
Hashtags & trending	Yes	Yes	Yes	Yes	Yes	No (limited)	Yes	Yes
Stories / 24-hour posts	Yes	No	Yes	Yes (limited/flag)	No (removed)	Yes (core)	No	No
Creator monetization/ads manager	Yes	Yes	Yes	Yes	Yes	Partial	Yes	No
Advanced analytics dashboards	Yes	Yes	Yes	Yes	Yes (Studio)	Partial	Yes	No
Groups/communities	Yes	Yes (Communities)	No	No	No (channels, not groups)	No	Yes	Yes
Auto-expire posts (TTL)	No	No	No	No	No	No	No	Yes (built-in)

Pseudonymous sign-up	No (real-name policy)	Yes	Yes	Yes	Yes (channel alias)	Yes	No (real-name oriented)	Yes
Inline fact-check pill (general claims)	Yes (but unstructured)	Yes (limited coverage)	No	No	Yes (context banners; limited)	No	No	Yes (verdict + confidence)
Author trust score visible	No	No	No	No	No	No	No	Yes (Trust Badge)
Link previews	Yes	Yes	Yes	No (limited)	N/A (video desc)	No	Yes	Yes (server-side unfurl)
Discovery de-prioritization via trust	No	No	No	No	No	No	No	Yes (gentle ranking)
Verified identity / blue badge	Yes	Yes	Yes	Yes	Yes	Partial	Yes	No

2.4 Summary

This background chapter explains why Mask App focuses on two main feature, time and trust-to rethink social media use. Studies show that endless feeds and old posts resurfacing keep people scrolling without pause, which wastes attention. Mask addresses this with post expiry (TTL), so content naturally disappears after a set time unless it gets enough reactions. This creates a stopping point for users but still lets them share freely.

Research on misinformation also shows that outright removal of posts is not the best answer. What helps more is context-short notes attached to claims and signals about the author's past credibility. Mask uses this approach with fact-check pills on posts and Trust Badges for authors. These signals let users judge content for themselves, making the platform feel transparent instead of restrictive.

When comparing to larger platforms, it's clear they excel at content creation and discovery but rarely show credibility signals or finite feeds. Mask keeps the familiar parts-profiles, posts, groups, pages, messages, notifications-but adds time limits and trust indicators to make the feed healthier. At this stage, heavy features like ad systems or advanced editing were left out so the app stays lean, easy to deploy, and manageable for a student project. These choices shaped the requirements and design that follow, where retention rules, fact-check triggers, and trust scoring are shown as working features. The overall aim is balance-keeping Mask usable and private while adding credibility and healthier ways of engaging with the feed.

Chapter 3

Research Methodology

This chapter explains how the Mask App was specified, designed, and implemented. It covers the requirement analysis, design specifications, detailed methodology, project plan, and task allocation.

3.1 Requirement Analysis & Design Specification

The following section focuses on the analysis of Mask App's functional requirements, outlining how the design meets user needs through clear specifications. It highlights core capabilities such as post creation with TTL, reactions, comments, groups and pages, encrypted messaging, fact-check integration, and scoped moderation, all tied together in a MERN-based architecture. These choices were made to keep the app user-friendly, efficient, and scalable while supporting privacy and fairness across the platform.

3.1.1 Overview

The way we built Mask App was very hands-on and based on real evidence, not just theory. Since we were a small student team, we followed an Agile approach that let us move in small steps and adjust as we learned. We started by looking at the social apps people already use every day and picked out the features that felt essential-things like posting, reacting, commenting, sharing, messaging, groups and pages, search, and notifications. At the same time, we read up on attention economics, misinformation, moderation, and private messaging to see where most platforms go wrong and what simple nudges could actually help. Instead of keeping these as ideas on slides, we jumped straight into working code: spinning up prototypes early, setting up environment variables, fighting with CORS errors, and testing encryption paths. That way, the design grew out of real constraints and problems we faced directly, not just wishful planning.

We picked Agile because we knew from the start that the scope would keep changing as we learned more. Instead of long planning cycles, we worked in short weekly sprints, mixing a bit of Scrum and Kanban to keep things moving. We kept one simple backlog that we constantly updated, so priorities shifted as soon as we discovered something new. Each feature was built as a full slice from start to finish-UI, API, database, and sometimes even the worker or WebSocket-so we could show it working right away and get feedback fast. Our acceptance notes were written in plain user terms, like making sure a reaction really triggers a notification with the proper

"reaction" type. That small detail helped us catch the Enum mismatch early without slowing down other tasks.

As we kept working, two clear guardrails slowly took shape that really define the app. The first was post retention (TTL), which makes sure very old posts drop out of the feed so people don't get stuck in endless scrolling. The second was credibility signals- adding a small verdict pill on claims and an author Trust Badge that updates based on their history. To make all this possible we stuck with the MERN stack, because it gave us speed and a single language across the whole project. React on Vercel runs the client, Node/Express on Render runs the API, and MongoDB Atlas handles storage. We leaned on environment configs so dev and production stayed aligned without messy changes. Our process was always about small steps: shipping little slices, checking with users quickly, and putting in simple budgets and rate limits for the fact-check worker so costs wouldn't get out of hand. We also kept accessibility and privacy in mind from the start, which is why direct messages are stored with AES-GCM encryption instead of just plain text.

Practically, this Agile approach reduced risk, made progress visible, and kept momentum high. Issues like the admin 403 and deployment wiring were surfaced by working builds, addressed in the next iteration, and verified with concrete demos rather than long plans. The result is a credibility-aware, time-bounded social feed that is realistic to operate by a small team and ready to evolve as new evidence and user feedback arrive.

3.1.2 System Design

The system architecture of Mask App has been designed as a layered structure that brings together the client, the application server, and the database, with selective integration of third-party services to extend functionality (Figure 3.2). At the client layer, users access the platform through a React web interface deployed on Vercel, which communicates securely with the backend API. The heart of the system is the Node/Express application server, which handles business logic such as post creation, comment and reaction tracking, message encryption and decryption, and fact-check processing.

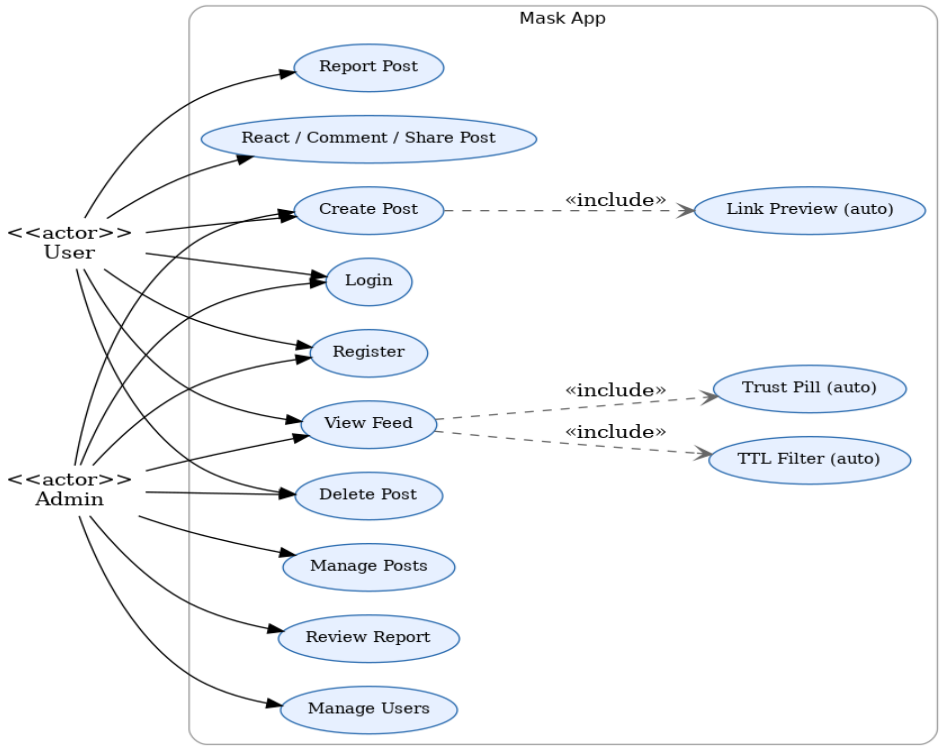


Figure 3.1: Mask Use Case Diagram

Additional services are tightly integrated into the application layer to make the system richer and more reliable. A dedicated link-unfurl service automatically inspects any URL shared in a post and generates a clean preview with the title, description, and image, so users can understand the content at a glance without leaving the app. In parallel, the fact-check pipeline connects to an external API (Google Gemini), which analyzes claims and returns a structured verdict with an explanation and confidence score. These results are then aggregated into the author's Trust Badge, giving readers a clear, interpretable signal about the credibility of a user's past contributions.

For handling media, Cloudinary integration ensures that profile pictures and post images are stored and delivered efficiently through a global CDN. Instead of storing bulky files directly in the database, the system saves lightweight links and metadata, which reduces storage overhead, speeds up retrieval, and provides on-the-fly transformations such as resizing or format conversion. This design choice keeps performance high even as the volume of media grows.

The application server also acts as the gatekeeper for authentication and authorization. Using JWT-based sessions and bcrypt-hashed passwords, it protects user data while enforcing role-based access. Critical moderation rules are applied here as well: content can only be deleted by its owner, a global admin, or the

respective group/page admin, ensuring fairness and accountability across the platform.

All long-term data is stored in MongoDB Atlas, which serves as the backbone of persistence. Collections are organized around users, posts, messages, notifications, groups, and fact-check results, with indexes tuned for recency so that fresh posts and updates appear quickly in the feed. The database design supports scalability, allowing Mask App to handle larger communities without compromising speed.

Together, these services create a modular, service-oriented architecture that keeps the system flexible for future growth. Whether it is adding multilingual fact-check support, more advanced admin analytics, or new media formats, the platform can evolve without rewriting the foundation. This approach ensures that Mask App is not only functional for today's needs but also prepared for the demands of tomorrow.

3.1.3 Component & Service Architecture

The architecture of Mask App has been designed in layers so that each part of the system does its job clearly and connects smoothly with the others. On the surface, users interact with the frontend, which is built with React and deployed on Vercel. This is where they see the feed, create posts, react, comment, send messages, or manage groups and pages. Features like the PostCard, ReactionBar, and LinkPreviewCard keep the interface simple and familiar, while the Trust Badge and fact-check pill add credibility signals directly into the feed.

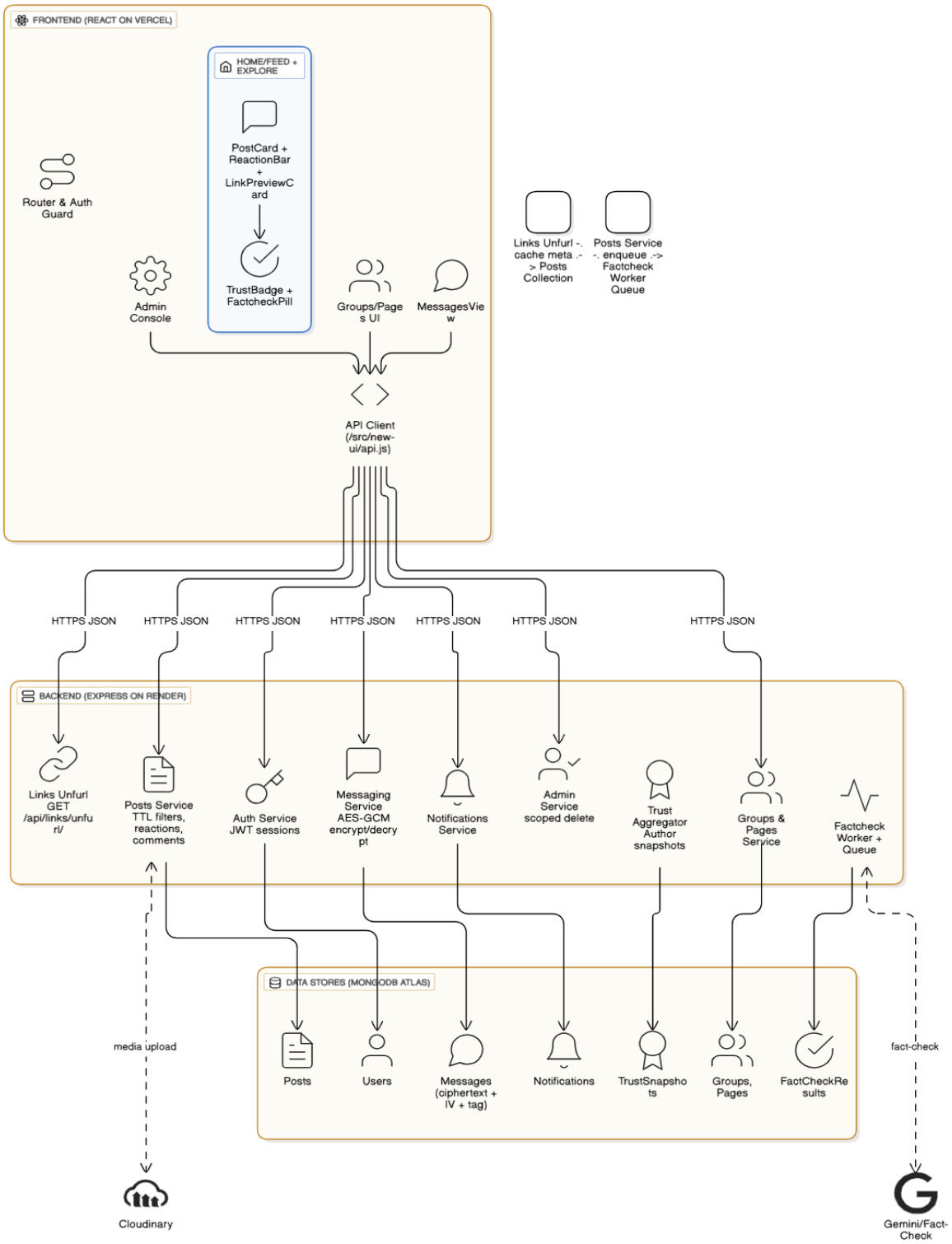


Figure 3.2: Mask App - Component & Service Architecture

Figure 3.2: Mask App-Component & Service Architecture shows how these frontend actions flow into the backend, where the real logic happens. The backend, running on Express and deployed to Render, takes care of post expiry filters, reactions and comments, notifications, and user authentication through JWT sessions. It also protects private conversations with AES-GCM message encryption and enforces moderation rules, so only owners, global admins, or group/page admins can delete content. Alongside these, special services enrich posts with link previews and manage fact-check requests through a queue system that connects to the external Gemini API.

All data is kept safe in MongoDB Atlas, which stores users, posts, messages, notifications, groups, and fact-check results in separate collections. The database is tuned to prioritize freshness, so new posts appear quickly while expired ones drop out automatically. Media files such as profile pictures and post images are stored in Cloudinary, while only their links are saved in the database for efficiency.

By combining these layers, the architecture stays clean and flexible. The frontend focuses on usability, the backend on reliable logic and security, and the database on safe storage and fast retrieval. Extra services like Cloudinary and Gemini extend the system without making it heavier. This design makes Mask App both practical to run today and ready to grow in the future, with room for features like multilingual fact-checks, richer admin analytics, and expanded media support.

3.1.4 Functional and Nonfunctional Requirements

This section outlines the functional and non-functional requirements of Mask App. The functional requirements capture the core features that make the system work-such as posting, reactions, comments, groups, encrypted messaging, fact-check checks, and scoped moderation-while the non-functional requirements focus on qualities like usability, performance, security, and reliability. Together, they ensure the app is not only feature-rich but also safe, efficient, and practical for everyday use.

Functional Requirements

Registration & Authentication

a. Users sign up and sign in; sessions use JWT.

b. Passwords are hashed (bcrypt or equivalent).

c. Pseudonymous profiles are allowed.

Profiles & Social Graph

a. View and edit profile (avatar, display name, bio).

b. Follow/Unfollow, basic suggestions.

Posting & Interactions

a. Create text/image posts; edit/delete own posts.

b. React, comment, share, bookmark.

c. Retention/TTL: posts expire automatically after a configured period; expired items are excluded from feed queries.

Groups & Pages (Scoped Visibility)

a. Create/join groups; create/administer pages.

b. Visibility rules: page posts visible to followers; group posts visible to members (private groups restrict resharing).

Direct Messages (DMs)

a. 1:1 messaging stored encrypted using AES-GCM with a base64 32-byte MESSAGE_ENC_KEY.

b. Decryption available only to authorized endpoints; IV and auth tag stored safely.

Link Previews

a. Server-side unfurl: GET /api/links/unfurl extracts title, description, and image with timeouts and basic sanitization.

b. Postcard renders LinkPreviewCard when URLs are detected.

Fact-check & Trust Signals

a. Server enqueues checks for candidate posts (engagement triggers or optional tag).

b. External model returns { verdict, explanation, confidence }.

c. UI shows a compact context pill on the post.

d. Trust Badge aggregates author history to a simple tier; discovery can gently de-prioritize low-trust authors.

Admin & Moderation

a. Delete authorization: owner OR global admin OR page/group admin.

b. Basic activity logs for sensitive actions.

Search, Notifications & Trending

a. Keyword search across users/posts.

b. Push/pull notifications for mentions, reactions, and follows.

c. Trending considers freshness and trust signals.

Deployment & Configuration

a. Frontend uses REACT_APP_API_URL (no trailing slash).
b. Backend reads MONGO_URI, JWT_SECRET, MESSAGE_ENC_KEY, optional Cloudinary credentials, and trust/fact-check budgets.

Non-functional Requirements

Performance

a. The system should respond to most common requests fast, with API calls ideally staying under half a second.
b. Tasks that are heavier like link unfurling or fact-checking are handled in the background so they don't slow down normal use.

Security & Privacy

a. All traffic goes through HTTPS/TLS and user sessions are secured with JWT.
b. Direct messages are stored using AES-GCM encryption so even in the database they are not in plain text.
c. Strict CORS rules, input validation, and rate limiting are applied to stop unauthorized access.

Reliability

a. The system should work even if the external fact-check service is not available; in that case posts just show no label instead of breaking.
b. Failures in one part of the system should not cause the rest of the app to stop working.

Scalability

a. The backend is stateless and runs with MongoDB, so it can be scaled horizontally when needed.
b. Features like links, trust signals, and messaging are isolated in their own modules to make scaling easier in the future.

Usability & Accessibility

a. Posting and interacting should be possible in very few clicks and expired posts must show a clear status.
b. The interface supports keyboard navigation and alt text for images to keep it more accessible.

Maintainability

a. The project has a clear folder structure (frontend in /src/new-ui, backend in /routes, /models, /services) which makes changes easier.
b. Environment variables are documented in .env.example and code comments help explain how the keys are used.

Portability

a. The system can run both locally and in the cloud without code changes.
b. Later on, CI/CD can be added to automate deployment, but the setup is already kept simple enough for student use.

By meeting these nonfunctional requirements, Mask App aims to balance smooth performance, strong privacy, and ease of use, while also making sure the system can grow and stay manageable in the future.

3.1.5 Data Flow Diagram

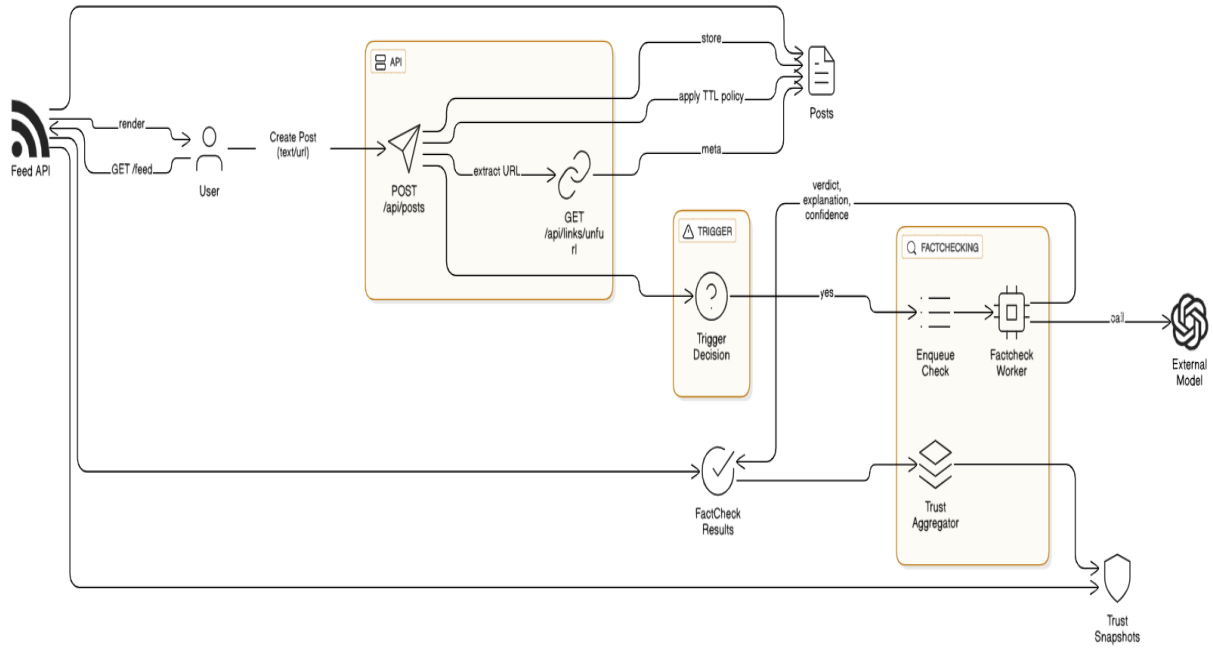


Figure 3.3: Data Flow Post Creation - Link Unfurl - Trigger Decision - Fact-check - Trust - Feed Render.

When a user creates a post, several background processes start working to keep the feed clean and trustworthy. After submission, the backend stores the post and applies its time-to-live (TTL) rule so that older content naturally expires. If the post contains a link, the system automatically calls the link-unfurl service to generate a neat preview with the page title, description, and image, saving users from opening extra tabs. As shown in Figure 3.4: Data Flow – Post Creation, Link Unfurl, Fact-check, and Trust Integration, the system also decides whether the post should be checked for credibility. This trigger can come from engagement thresholds or a #verify tag added by the author. Once triggered, the post is queued for fact-checking, and the external Gemini model returns a verdict, an explanation, and a confidence score. That outcome is displayed on the post as a compact fact-check pill and also recorded for the author, gradually building their Trust Badge over time. In this way, Mask App keeps the feed fresh while giving readers quick, transparent signals about reliability without hiding content.

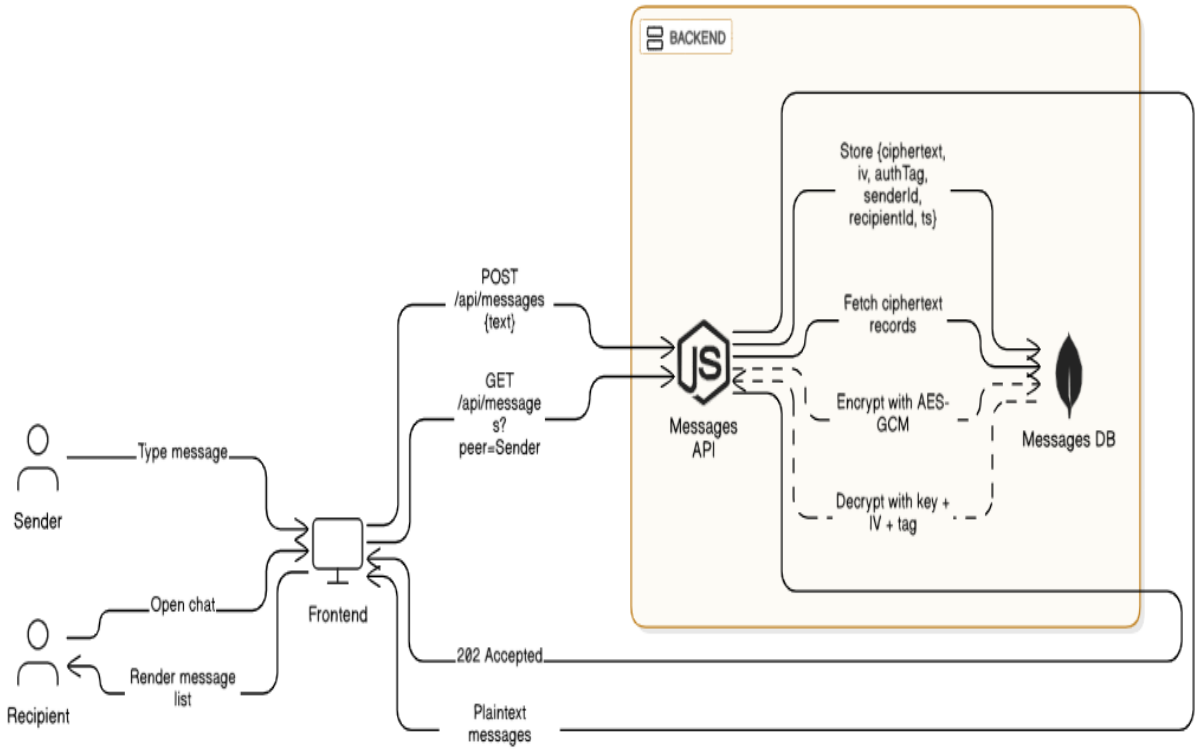


Figure 3.4: Data Flow - Direct Messages Stored with AES-GCM at Rest.

Private messaging is handled with the same care for both usability and privacy. When a sender types a message, the text passes through the frontend to the Messages API in the backend, where it is encrypted before being stored. The encryption uses AES-GCM, a modern method that not only converts the message into unreadable ciphertext but also produces an initialization vector and authentication tag. These pieces together ensure that the message can only be decrypted correctly and has not been tampered with. As shown in Figure 3.8: Data Flow – Direct Messages Stored with AES-GCM at Rest, the recipient can later open their chat, and the system retrieves the ciphertext from the database, decrypts it with the correct key, and renders it in plain text inside the conversation. Importantly, the database never stores raw messages, so even if someone had direct access to it, all they would see would be encrypted records. This design strikes a healthy balance: users enjoy simple, chat-like messaging while knowing that their conversations are protected at rest. It shows that Mask App takes privacy seriously even as a student project, and that security is built in from the ground up rather than added as an afterthought.

3.1.6 Entity-Relationship (ER) Diagram

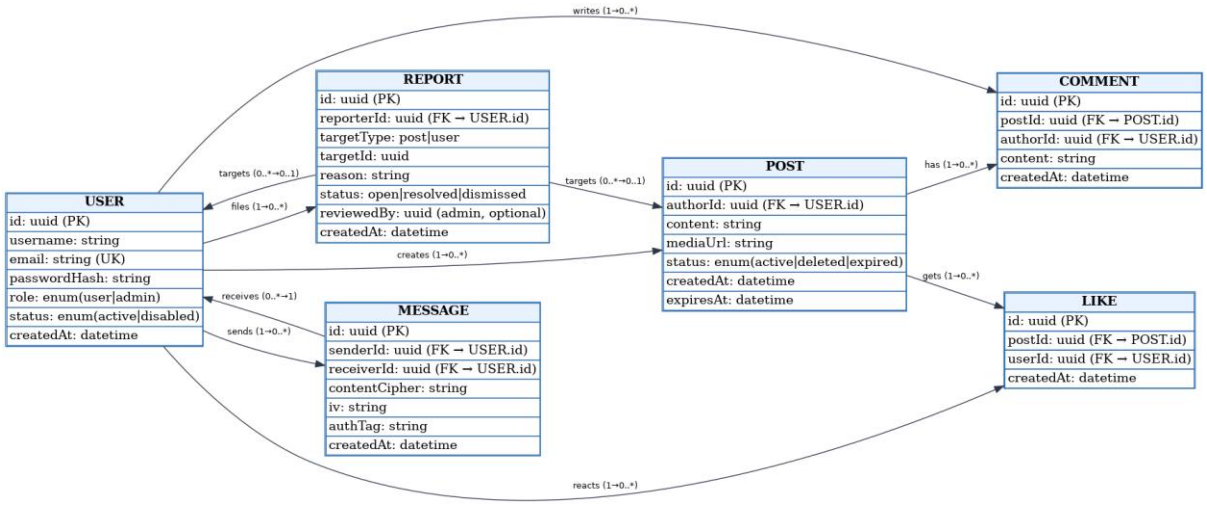


Figure 3.5: ER diagram of User Domain

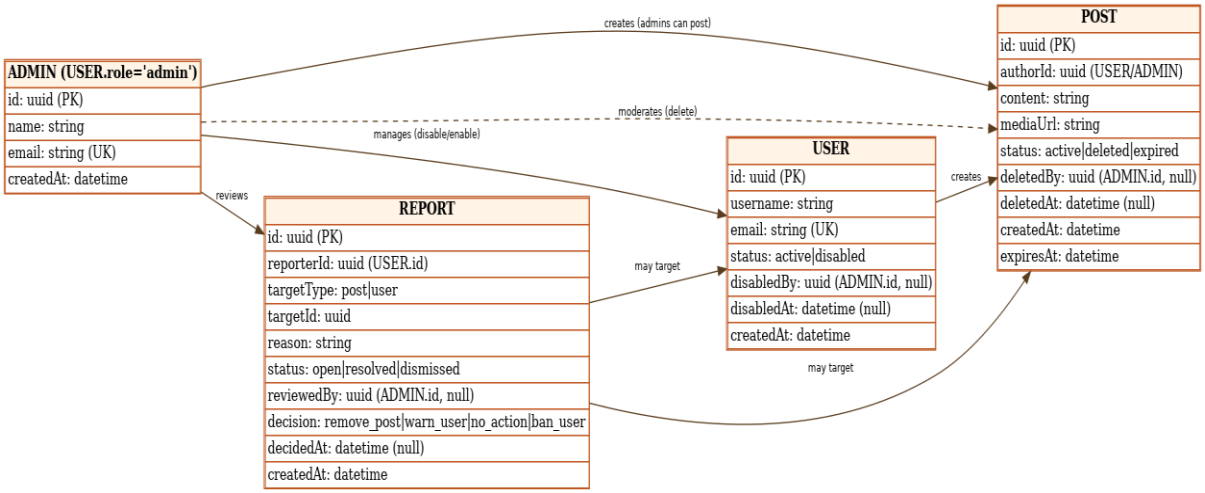


Figure 3.6: ER diagram of Admin Domain

Short form => Full form

PK => Primary Key

UK => Unique Key (unique constraint)

FK => Foreign Key (reference to another table/entity)

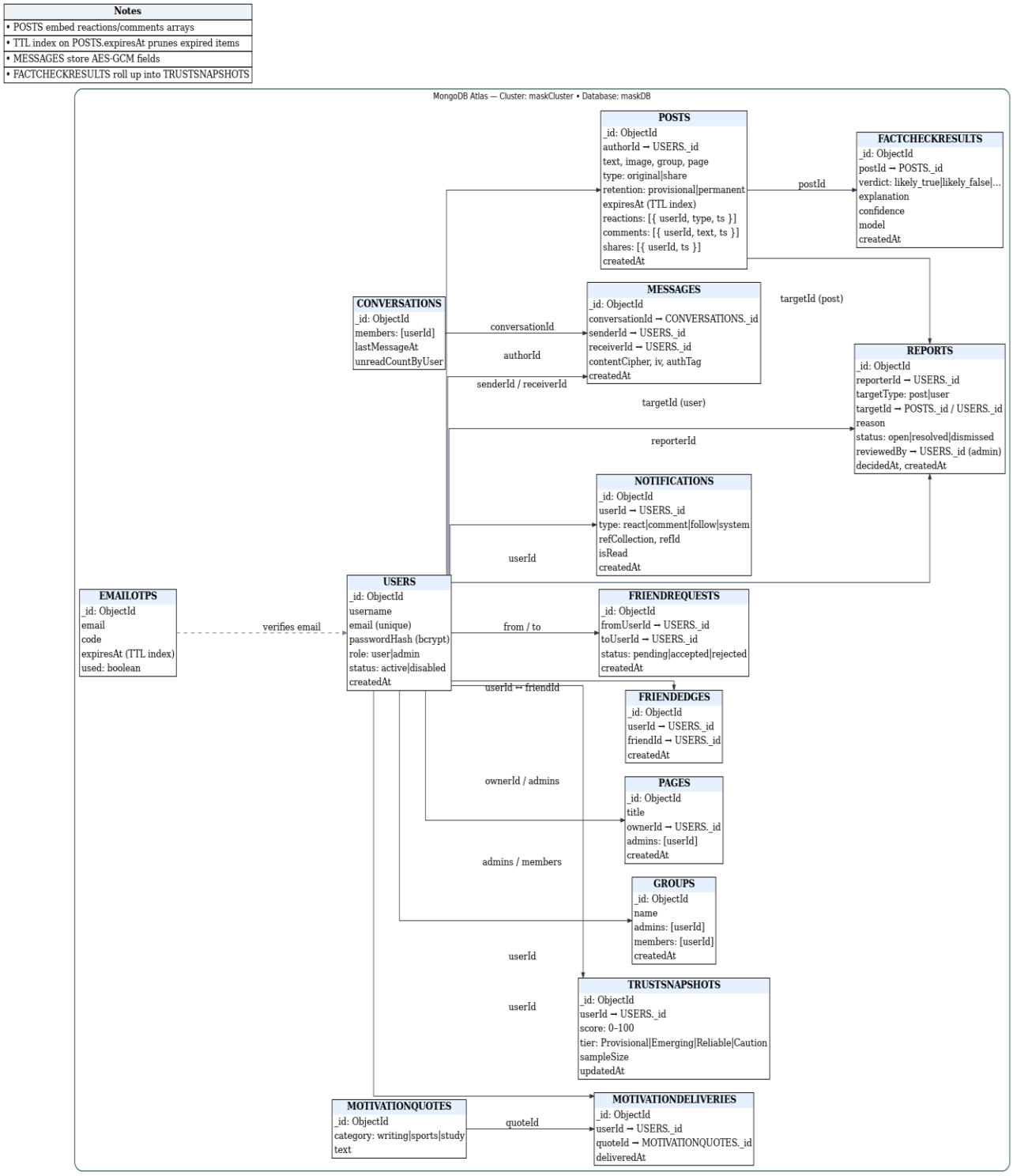


Figure 3.7: MaskDB Atlas Diagram

Figure 3.5: ER Diagram of User Domain focuses on the basic social interactions within the app. It defines how users create posts, comment, like, and report content, as well as how private messages are exchanged. Each user is identified by a unique ID, with attributes such as username, email, and role (user or admin). Posts are linked to their authors and can include content, media, a status (active, deleted, expired), and an expiry date for TTL enforcement. Reactions and comments connect back to both posts and users, ensuring that engagement history is stored consistently. Messages are stored securely, using encrypted content fields along with the IV and authentication tag required for AES-GCM decryption. Reports link users to specific posts or profiles, documenting the reason and outcome, which supports the moderation workflow. Together, these relationships form the foundation for everyday interactions in Mask App.

Figure 3.6: ER Diagram of Admin Domain extends this by zooming in on the responsibilities of administrators. Here, admins are modeled as users with elevated roles and additional authority. They can review reports, decide whether to remove content, warn users, or in serious cases, ban accounts. Admins can also disable or re-enable user accounts and delete posts, with all decisions stored as part of the report entity for accountability. This diagram highlights how moderation is enforced not only by logic in the backend but also by explicit relationships in the database schema. By keeping track of reviewer IDs, decisions, and timestamps, the system creates a transparent trail of actions that balances fairness and responsibility in community management.

Figure 3.7: MaskDB Atlas Diagram shows the complete cluster view of the MongoDB schema used in production. It brings together all collections: users, posts, groups, pages, messages, notifications, reports, fact-check results, and trust snapshots. Posts embed comments and reactions as arrays for quick retrieval, while TTL indexes automatically remove expired items. Messages are stored with AES-GCM fields (ciphertext, IV, tag), ensuring that even in the raw database, private content remains unreadable. Fact-check results are tied to posts and later aggregated into trust snapshots, which produce the visible Trust Badge tiers for authors. Other collections handle friend requests, friendships, groups, and pages, giving the system the flexibility of both open networks and smaller communities. There are also dedicated collections for motivational quotes and their deliveries, used to power the daily motivation feature. By centralizing all of these entities in MongoDB Atlas, Mask App benefits from a schema that is flexible enough for rapid changes but still structured to keep relationships consistent and queries efficient.

Overall, the three ER diagrams show how Mask App balances user interactions, admin governance, and system-wide storage in one coherent data model. Everyday activity-posting, reacting, messaging-is seamlessly tied to admin oversight and fact-

checking, while Atlas ensures everything is stored securely and kept fresh with TTL rules. This layered approach makes the system not only functional today but also scalable and adaptable for future features.

3.1.7 UI Design

The User Interface (UI) of Mask App is intentionally clean, consistent, and distraction-free so that visitors, signed-in users, group/page admins, and global admins can move through the product without friction. A neutral, low-contrast base palette keeps the focus on content, while a single accent color is reserved for primary actions to reduce cognitive load. Navigation is simple-Feed, Create, Groups/Pages, Messages, Notifications, and Admin-each using clear labels and predictable layouts. The feed centers on a compact PostCard that surfaces author, time, and a subtle TTL indicator; when a link is present, a LinkPreviewCard expands with title, description, and image. Reactions, comments, and shares live in a unified ReactionBar, and credibility signals appear only when available: a small Trust Badge beside the author and an on-post fact-check pill with brief model notes. Messaging uses a tidy bubble layout with an unobtrusive lock icon to reflect AES-GCM protection. Reporting opens a lightweight modal with guided reasons and a preview. Admin views reuse the same components but reveal scoped controls (delete/disable) only when the user has permission, keeping the rest of the UI calm. Throughout, responsive spacing, keyboard reachability, readable type, and subtle micro-interactions make the interface feel quick, accessible, and easy to learn.

The sign-up flow keeps things simple while doing the basics right. New users pick a pseudonym (no real name required), provide an email, and set a password. As soon as the email is entered, the server sends a short one-time code to verify that the address is real; the form acknowledges this with “Code sent. Check your inbox.” and unlocks the Create account button only after a valid 6-digit code is supplied. Fields are clearly labeled, input states are obvious, and errors are written in plain language (e.g., “code expired,” “email already in use”). On success, the backend stores a bcrypt password hash, creates a neutral Provisional trust profile, and signs the user in so they land on Home without repeating the login step.

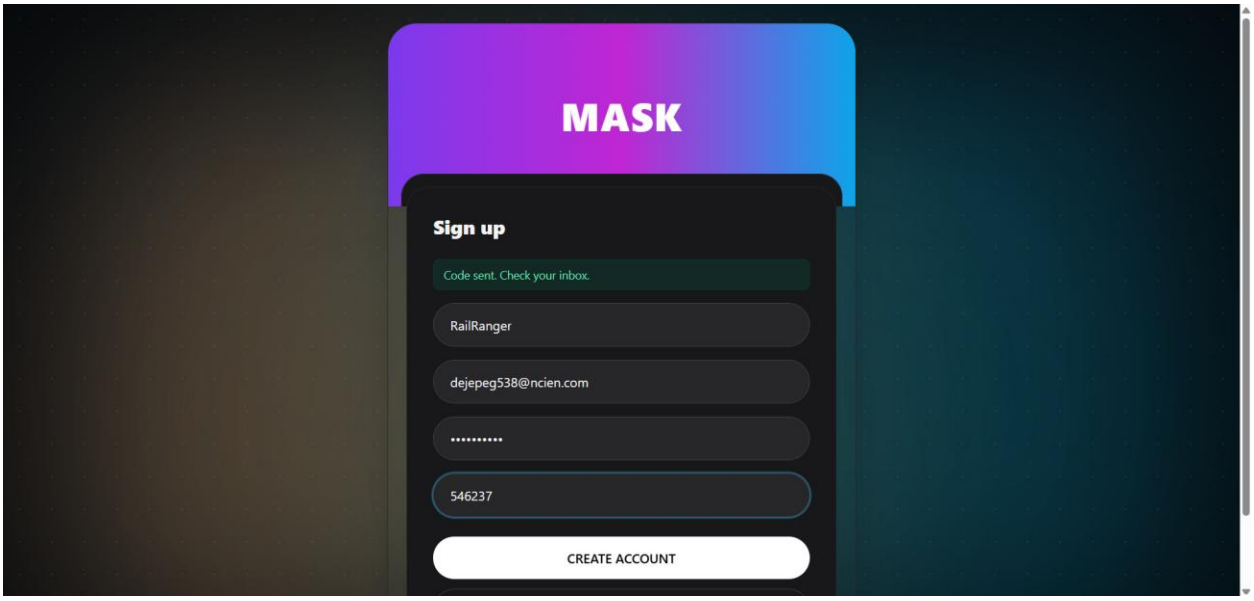


Figure 3.8: Sign-up page

The log-in card mirrors the same compact design: username and password, a clear Sign in action, and a secondary link to Create account. Failed attempts don't leak detail ("invalid credentials" rather than which field was wrong), and short lockouts apply after repeated misses to discourage guessing. When authentication passes, the server issues a secure, HttpOnly session cookie (JWT under the hood) with the correct SameSite settings for the deployed domains. From there, the app returns the user to where they left off-typically the Home feed-so they can pick up without friction.

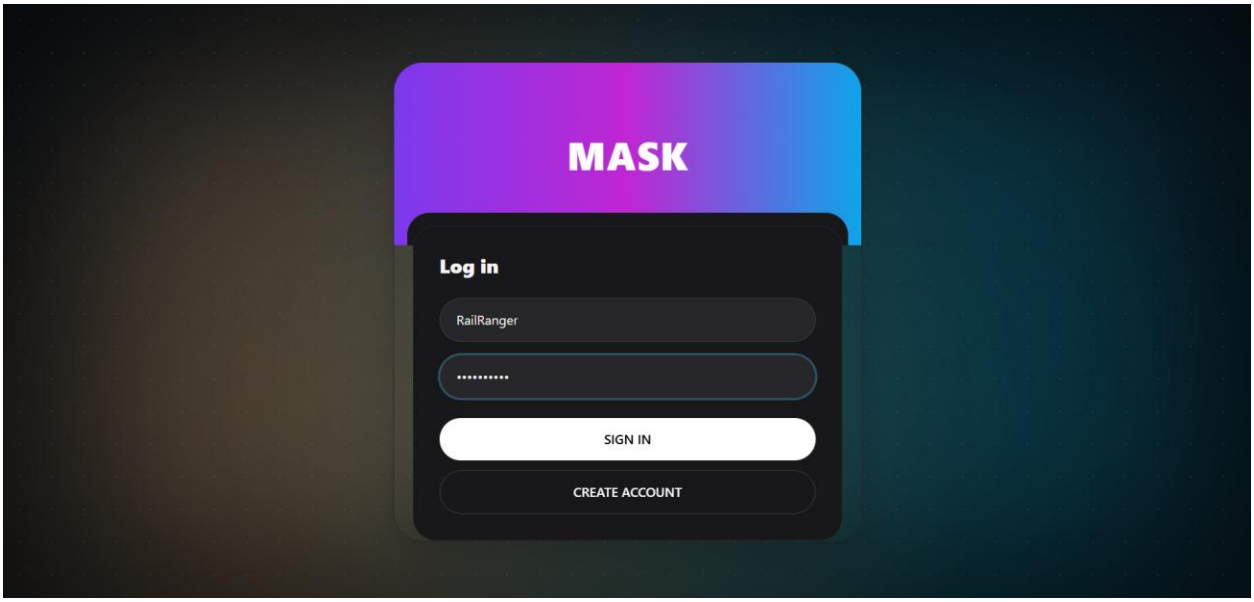


Figure 3.9: Log in page

Both pages use a focused, high-contrast card on a subdued background to keep attention on the task. Password rules are shown inline before submission, and the submit button stays disabled until the form is valid. Verification codes expire quickly and can be re-sent from the same screen. Sessions respect the project's time cap: you'll see well-being nudges during long sessions, and an automatic sign-out after the limit. All traffic runs over HTTPS; credentials never touch client storage; and CORS settings are pinned to the frontend origin. The result is a sign-up/log-in experience that's fast, privacy-respecting, and secure enough for a pseudonymous social app.

Homepage of Mask

The Mask home screen follows a familiar three-column layout-LeftNav, Middle feed, and RightRail. New users can orient themselves instantly.

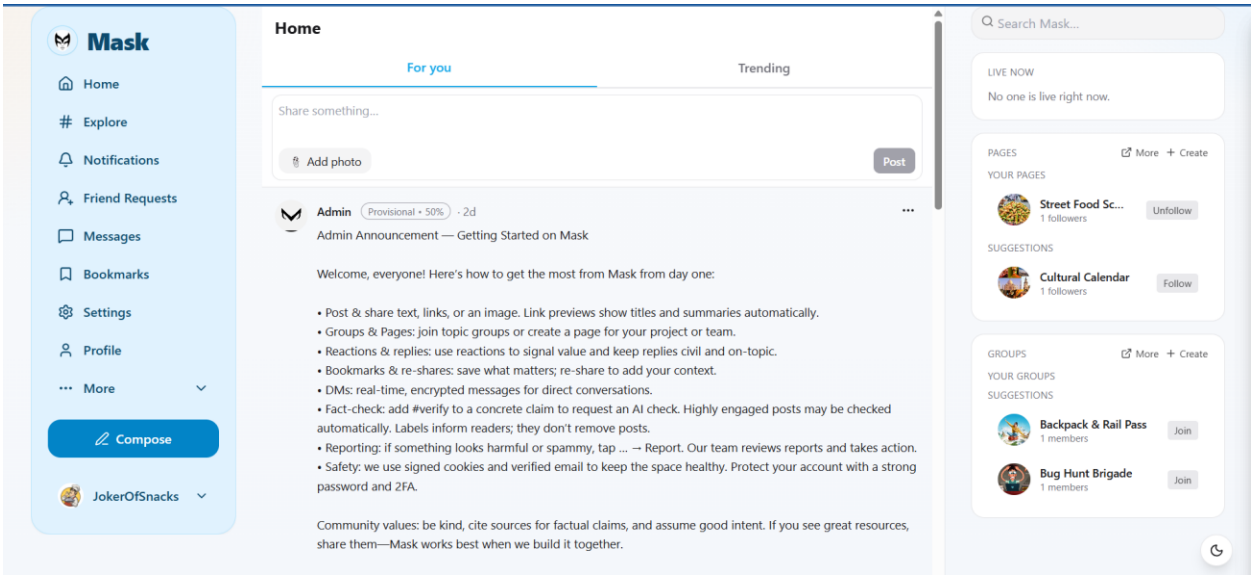


Figure 3.10: Homepage page

The LeftNav is a persistent vertical menu with clear icons and labels: Home, Explore, Notifications, Friend Requests, Messages, Bookmarks, Settings, Profile, and a "More" drawer for secondary items. A prominent Compose button sits above the account switcher, making it easy to start a post from anywhere without hunting through the UI.

The center column is the heart of the product. A lightweight composer ("Share something…") supports text and images, and a simple Post action keeps the flow fast. Below it, the feed presents PostCards with author, time, and the usual actions-react, comment, and share-kept together for muscle memory. Tabs let you switch context: For you and Trending on the Home view; choosing Explore swaps in a discovery feed,

selecting Notifications shows your updates, and Messages takes you to the chat experience. When a link is included, a link preview expands in place so the post stays readable without leaving the page.

The RightRail holds quick utilities that don't interrupt reading. A universal Search sits at the top; beneath it a small Live Now panel appears when a stream is active. Sections for Pages and Groups surface what you own and what's suggested, with one-click Create and More shortcuts for deeper browsing. A theme toggle sits at the bottom right so you can switch to dark mode without opening Settings. Overall, the page uses a soft, neutral palette with one accent color for primary actions, generous spacing, and consistent components so the content, not the chrome, stays in focus.

Auto-expire with engagement boosts

Mask uses a time-bounded feed by default: every post is created with a 24-hour lifespan. You can see that countdown as a small "time left" chip on the PostCard (e.g., "18h 0m left" in the first screenshot). Once the timer hits zero the post falls out of the feed automatically, keeping the experience fresh and preventing the endless-scroll effect.

To keep good content from disappearing too quickly, the timer adapts to engagement. We look at unique reactions (one reaction per account; switching emojis doesn't inflate the count). When a post crosses simple, public thresholds, its lifetime stretches:

a. **5 unique reactions** => the post graduates from the 24-hour window to a 7-day window. The "time left" chip updates immediately to reflect the new expiry (the second screenshot shows the post after it starts receiving reactions).

b. **7 unique reactions** => the post becomes permanent (no expiry). The chip disappears and the post is treated like any normal, non-expiring item (the third screenshot shows a post that's gathered more reactions and is now saved).

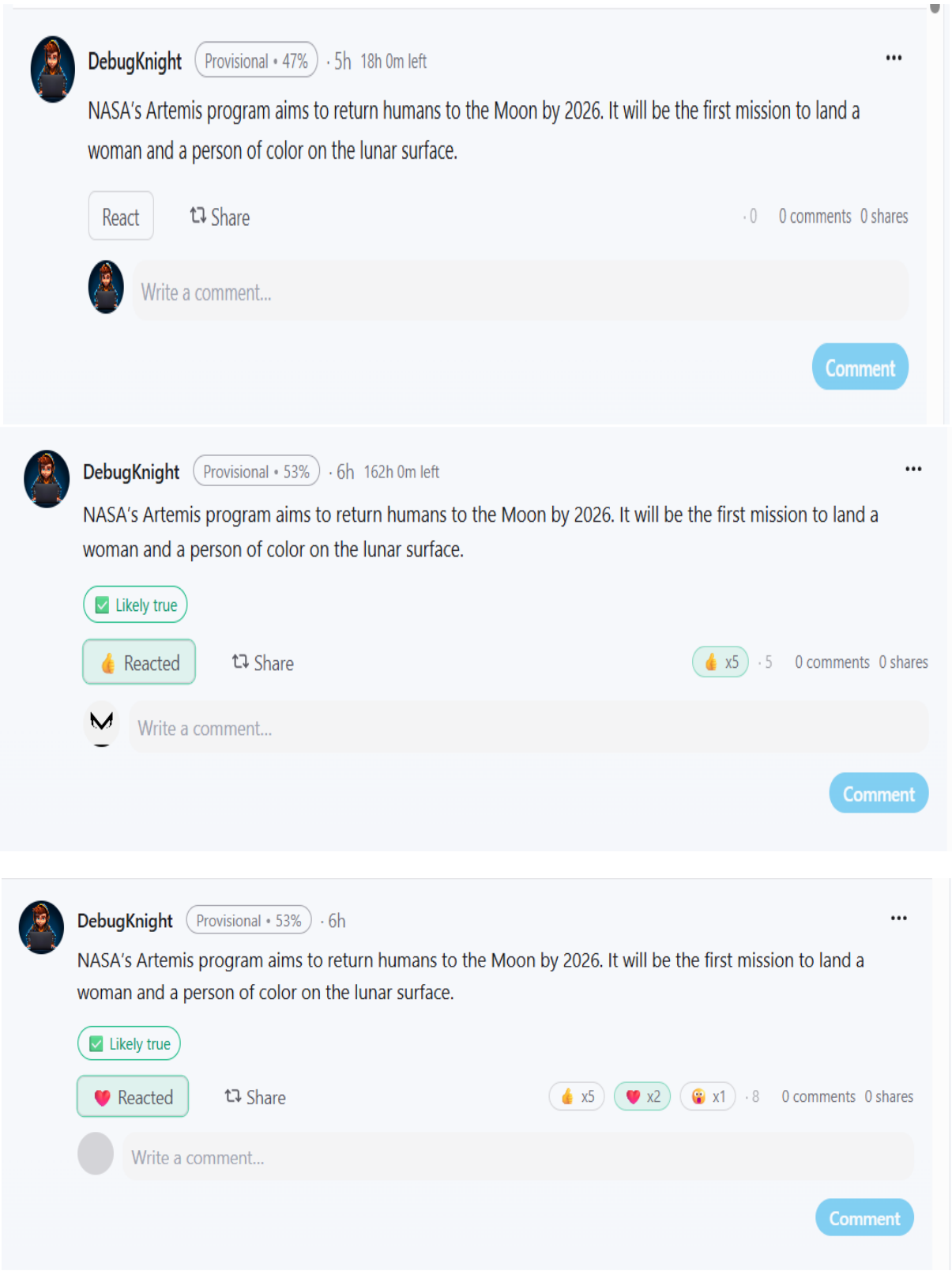


Figure 3.11: Auto-expire posts (TTL)

This scheme does a couple of practical things. It means the feed never looks empty on quiet days because high-quality posts naturally stick around, but new posts still get a fair shot because everything starts with the same 24-hour runway. It also makes the rule easy to understand for users: if people value a post, reacting to it literally gives it more life.

A few details under the hood keep things tidy and fair. Only distinct users are counted; a single person reacting five times still counts as one. Un-reacting removes that user from the tally, so a post can drop back from the 7-day tier to the 24-hour tier if interest fades (it won't revert from "permanent," though=>that crossing is sticky by design). Shares and comments don't affect the timer right now; they drive discovery and discussion but we keep the lifetime rule anchored to a simple signal. Admin deletes still win=>owners and admins can remove a post at any time regardless of its tier. Pinned posts are separate; pins are always visible in their context and don't use the TTL clock.

The server stores an expiresAt value when a post is created and updates it on each reaction event. If the post crosses the 5-reaction line, expiresAt moves forward to "now + 7 days" (or further if it already had more time). When it crosses 7 reactions, expiresAt is cleared to mark it as permanent. Expired items are filtered out at query time and also cleaned up by a background job, so feeds stay small and fast. For fairness across big and small communities, the thresholds can be tuned with environment variables; on smaller cohorts we can lower the numbers, and on busy cohorts we can raise them slightly so permanence really means "the crowd cares."

Put simply: every post gets a day to prove itself; the community can extend that to a week with just a handful of genuine reactions, and the very best posts can live on. The result is a feed that stays fresh without feeling thin, and a rule that users can see and trust.

Fact-check & context labels

Mask adds a light, readable layer of credibility signals to posts that look like factual claims. When a check runs, a small pill appears on the card=>Likely true or Likely false-with a confidence score. Clicking the pill opens a Post context panel that shows the exact claim the system evaluated, a short model note in plain language, and the model identifier. These labels are advisory; they help readers assess a statement. They do not hide or delete posts, and moderators can review or override any outcome.

Checks are requested in two ways. First, by engagement trigger: once a post attracts a small number of unique reactions, the system queues a single verification job for that post. For testing, the threshold is set low (two unique reactions) so the behavior

is easy to see; in production it can scale up or down with community size. Second, by author intent: adding the tag #verify requests an immediate check at creation time. To keep costs predictable on student budgets, the queue enforces a per-hour budget and a minimum spacing between calls. If we hit either limit, no label is shown—readers aren't distracted with partial results.

Because the feature uses an external model, the interface is designed to communicate uncertainty honestly. We avoid absolutes ("True/False") and use Likely phrasing with a visible confidence. The context panel includes the model name and a timestamped note so readers can see *what* the model believed and *when*. Those choices matter: models can lag reality, miss context, or simply be wrong. Moderators and admins always have the last word, and readers keep all normal actions (react, comment, share, report).

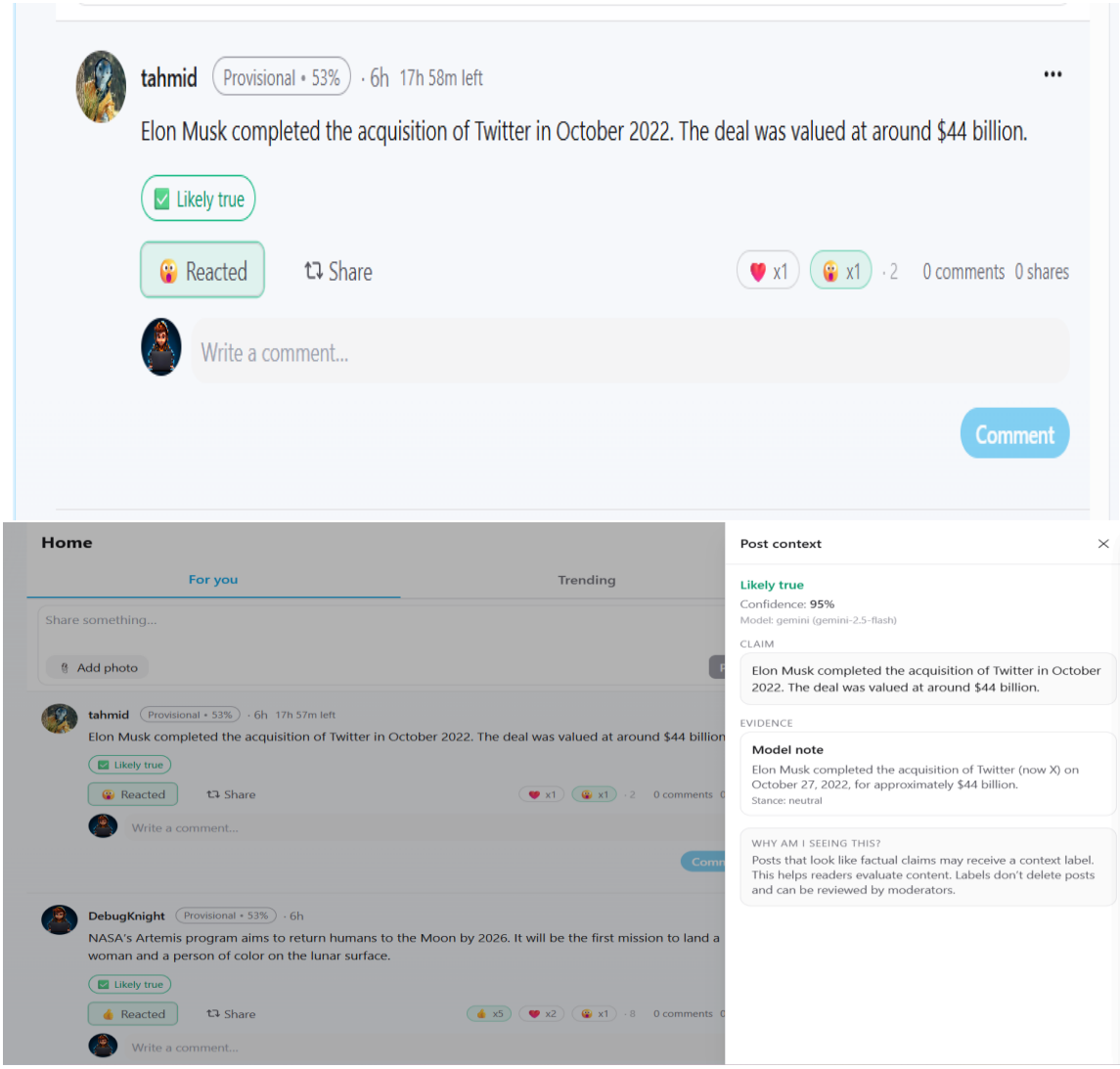


Figure3.12: Inline fact-check pill

Known limitations (and why they matter)

No automated system is perfect, and it's important to be clear about where this one can stumble. The most common risk is recency: models aren't updated minute-by-minute. If a leadership change or policy shift just happened, a claim may be judged using last month's information. The screenshots show this risk-one post about the U.S. presidency received a label that reflected a 2024 snapshot, even though the situation in 2025 had changed. That's not a user error; it's the predictable edge of using a time-bounded model.

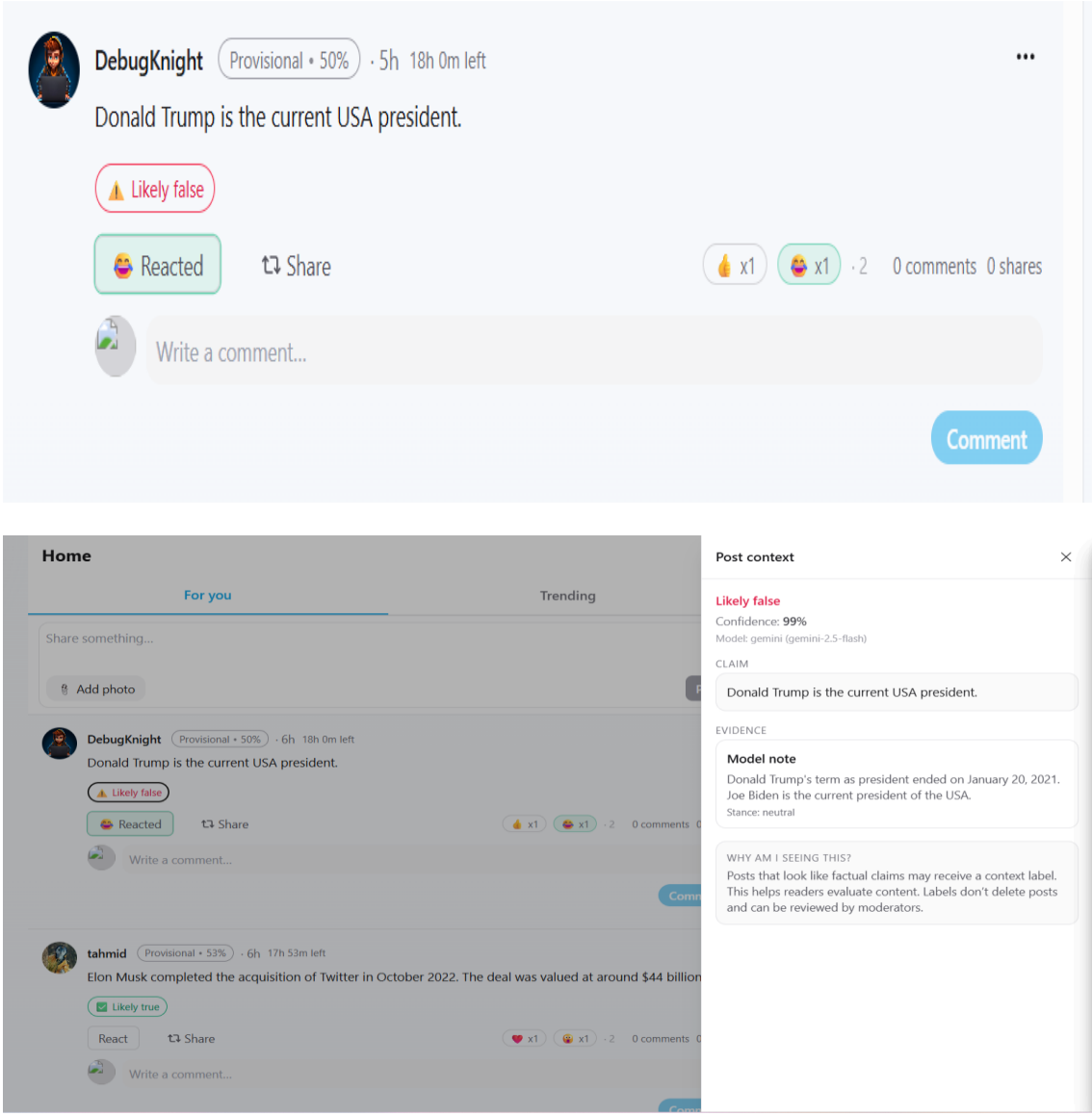


Figure 3.13: Limitations of the inline fact-check pill

Ambiguity is another source of error. Compound or underspecified claims ("X approved Y this year") can be partially true in one jurisdiction and false in another; sarcasm and jokes are easy for machines to misread. Multilingual and transliterated names sometimes confuse entity resolution, and paywalled or dead links make it harder for a model to validate context. In fast-moving events, evidence may be incomplete, so even a high confidence score can age quickly. Finally, practical limits apply: rate budgets mean not every post will be checked; if the queue is full or the budget is exhausted, the app simply shows no label rather than guessing.

How Mask mitigates these limits

The product is built to fail safe. Labels are context, not enforcement; nothing is removed automatically. The UI shows Likely wording and confidence instead of absolutes, along with the model name and note so readers can judge the source. Only unique users count toward the trigger (one account, one vote), which reduces the chance of brigading a label into existence. Moderators can re-check a post after major news or apply a manual note when a topic is time-sensitive. When a check is skipped due to budget or spacing, no label is shown at all. Longer term, we plan periodic rechecks for certain topics, multilingual support, and a small reviewer tool so edge cases can be corrected quickly with a clear audit trail.

In short, the feature is intentionally modest: it adds a nudge toward accuracy without getting in the way. Good claims earn a small green pill; dubious ones get a cautious red tag; everything else stays clean. And when reality changes faster than models do, the product tells you that plainly and leaves the decision with people.

Trust score & badge - full version

Mask allows pseudonyms, which is great for privacy but makes it harder to judge credibility at a glance. To bridge that gap, every author carries a small Trust Badge beside their name. It isn't a popularity meter and it isn't a ban switch-it's a quick, interpretable summary of how the author's recent factual posts have fared when we've run checks.

Here's the idea in plain terms. When a post receives a context label such as "Likely true" or "Likely false," that outcome is written to a compact record with the model's confidence. The trust service then updates the author's score using a conservative baseline (new accounts start neutral), weights each result by confidence, and applies time decay so older outcomes matter less. One strong result never defines a person; a pattern over time does. Scores are clipped to a sensible range and mapped to clear tiers you can read at a glance: for example, *Provisional* (new or little history), *Emerging*, *Reliable*, or *Caution*. The badge shows the tier label and a small percentage so readers get both a category and a number.

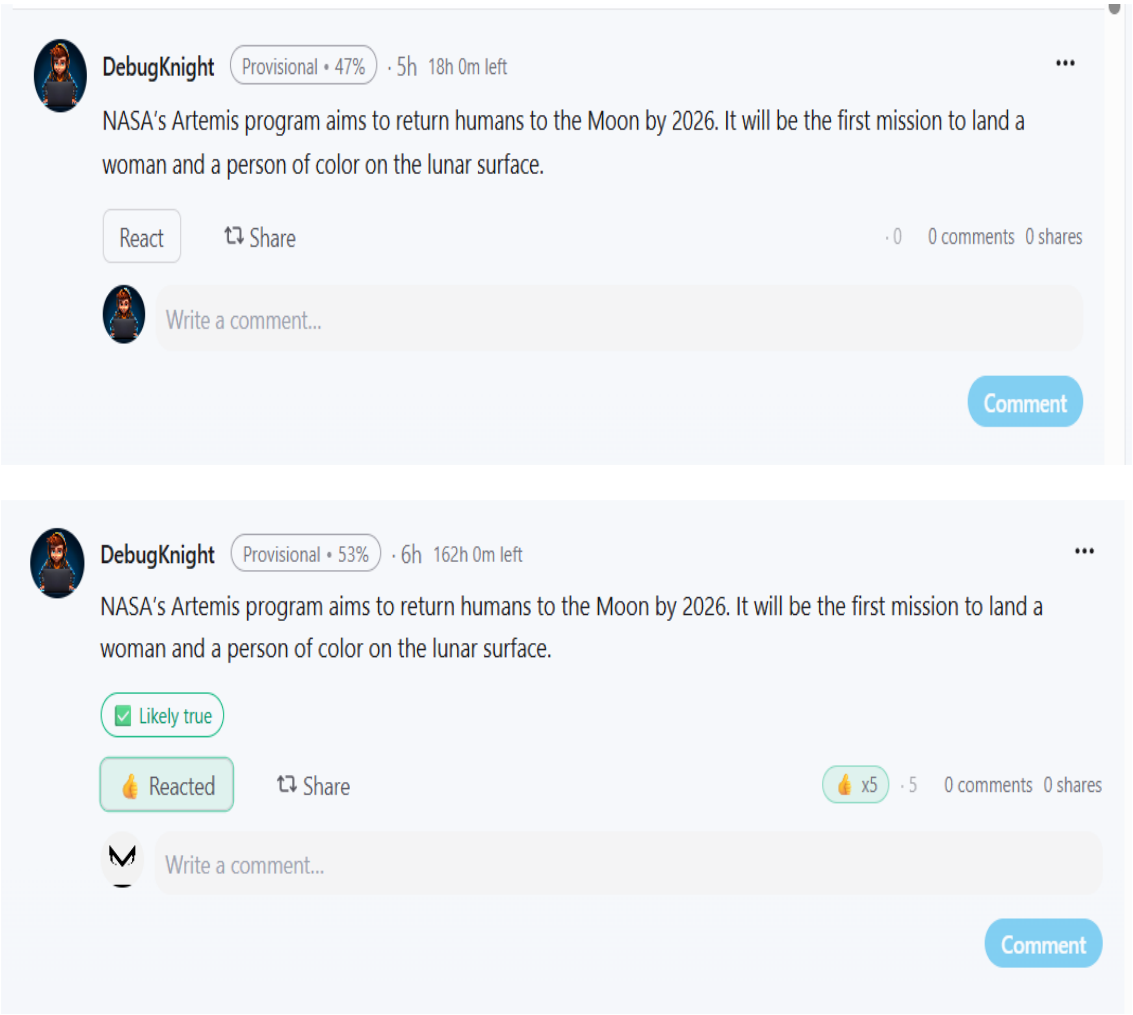


Figure 3.14: Author trust score visible

You'll notice the badge appears only when there's enough signal. If a post hasn't been checked-or the system skipped a check due to rate limits-we don't change the author's score. Likewise, a single mislabeled post won't tank someone's reputation: the aggregation uses a Bayesian-style prior and recency weighting so one-off errors or jokes don't dominate. When a claim is later corrected or re-checked, the snapshot is recomputed automatically. In the feed, the badge sits quietly next to the avatar; tapping it opens a short panel with a few recent outcomes so the score isn't a mystery.

Fairness guardrails are built in. Only unique users can push a post to a check (via engagement triggers or the #verify tag), which dampens brigading. A label never removes content; it adds context, and moderators can override if needed. The badge itself is advisory-it doesn't change who can post or comment. It can, however, inform ranking in discovery views such as Trending so untrustworthy authors don't get extra lift while reliable voices are a bit easier to find. New users aren't penalized: they start in the *Provisional* tier and move as their own posts accumulate history.

From a user's perspective, the benefit is simple: in a pseudonymous space you still have a quick, human-readable signal about an author's track record. From an operator's perspective, the implementation stays lightweight. Scores are stored as snapshots and updated when new fact-check results arrive, which keeps reads fast and costs low. If your community grows or norms change, thresholds and decay rates can be tuned without redesigning the system.

In short, the Trust Badge doesn't replace judgment; it gives you a small, honest head start. When a post you're reading shows "Likely true," that positive outcome nudges the author upward. When a claim is labeled "Likely false," it nudges them down. Over time, those nudges add up to a score that reflects behavior-not identity-and helps everyone read the feed with a little more confidence.

Well-being nudges & session cap

Mask builds "time awareness" into the feed so it's easier to step away. During a session you'll see two gentle nudges-a small card at 15 minutes and another at 30 minutes-that suggest a quick stretch or water break. They slide in near the post you're reading, use plain language, and dismiss with one tap so they don't derail your focus.

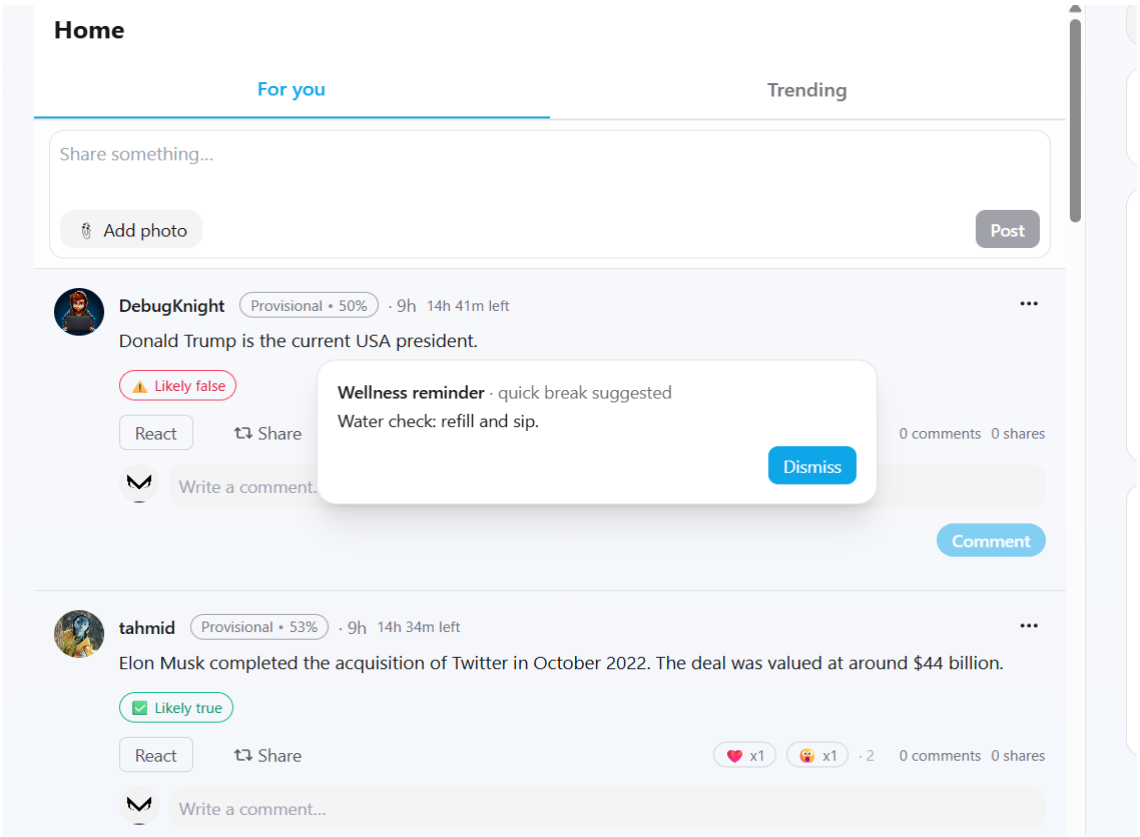


Figure 3.15: Wellbeing nudges

If you keep scrolling, a larger banner appears at 43 minutes with a clear heads-up that a session timeout is coming. The banner offers two choices: Keep reading (close the banner and continue) or Logout now (end the session immediately). At the 45-minute mark the app signs you out automatically; you can jump back in anytime by logging in again. This small bit of friction is intentional-it breaks the "just one more post" reflex without shaming the user or hiding content. Timers reset on re-login, so you're always in control.

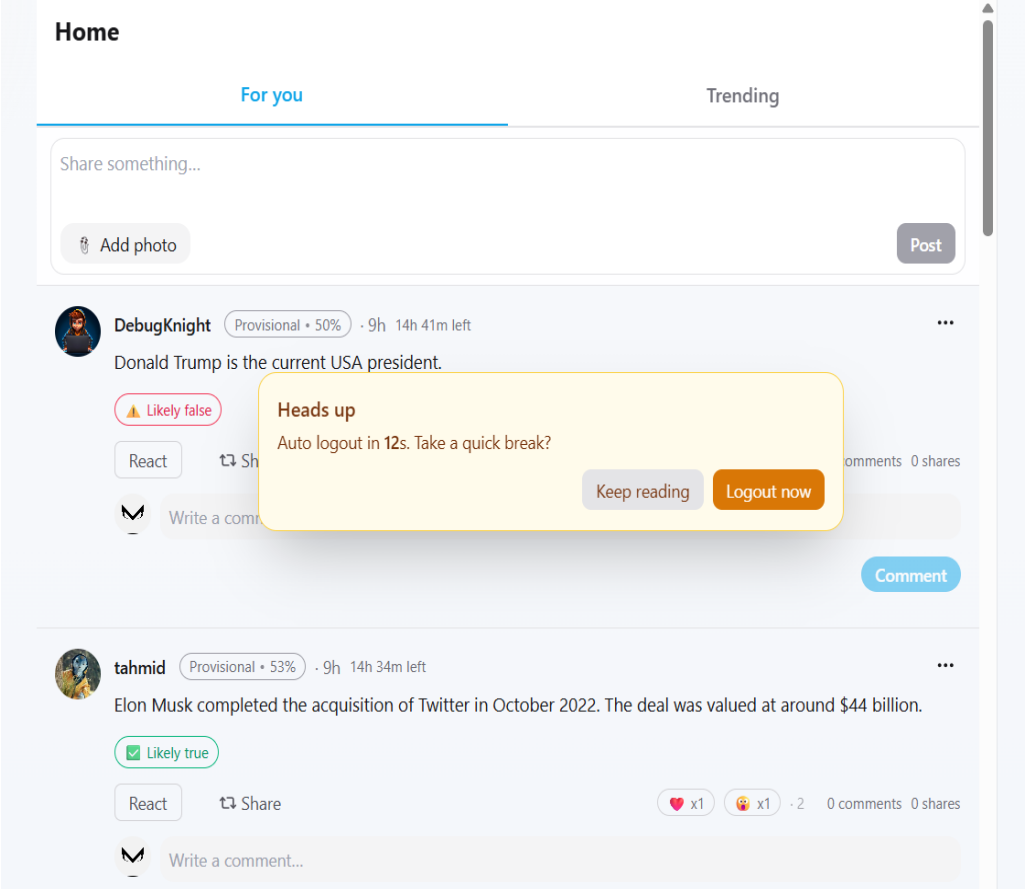


Figure 3.16: Forced Log Out

A couple of details keep the experience respectful. The nudges are keyboard-navigable and readable with high-contrast text. They don't cover content or block reactions and comments. If the tab is idle or in the background, the timer pauses; the goal is to curb passive scrolling, not penalize you for stepping away. We treat the feature as a default for healthier use, especially for students, but the design leaves room to make the intervals configurable later if your community needs it.

Daily motivation card

To start sessions on a positive note, Mask shows a daily motivation card right after login. The message isn't generic; it's picked from short sets that match what you've told the app about yourself-your goals, hobbies, or study interests in Settings. A writer might see a one-line prompt about getting words down, a football fan might get a training nudge, and a student preparing for exams might get a quick focus tip. Cards are short, quietly styled, and easy to dismiss. They're there to set intent, not to compete with the feed.

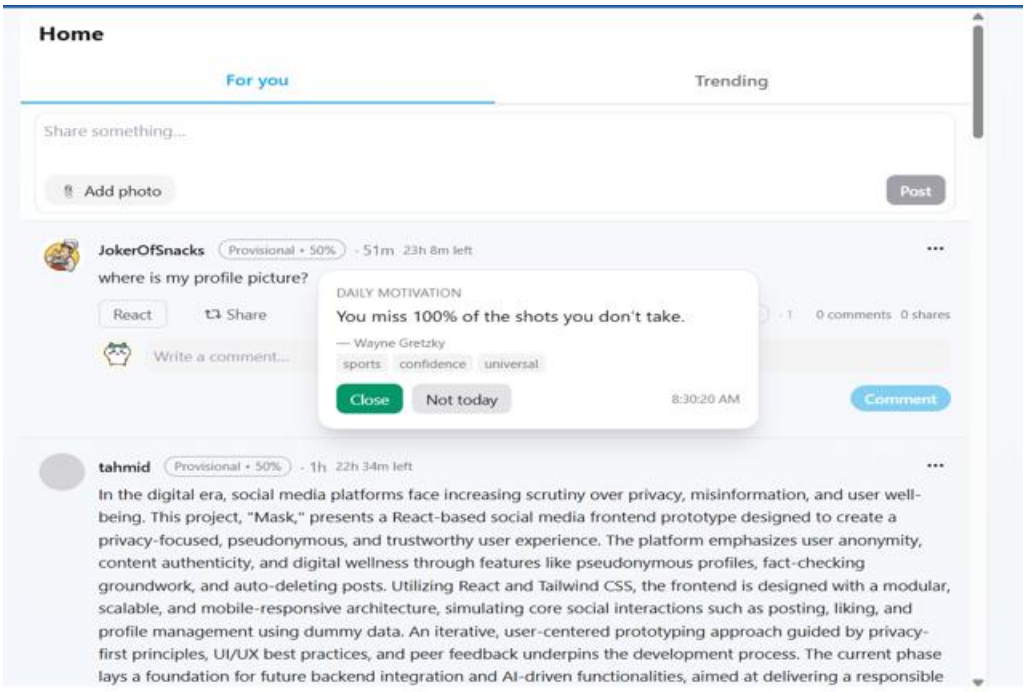


Figure 3.17: Daily Motivation Quote

Together, these two touches-lightweight nudges and a firm session cap, plus a small dose of personalized motivation—support the central purpose of Mask: a social space that respects your time and helps you leave feeling better, not drained.

3.2 Detailed Methodology and Design

Work proceeded in short iterations: scaffold => feature slice => test => refactor. Early sprints focused on the feed and retention logic; later sprints added groups/pages, messaging encryption, link unfurl, and finally fact-check/trust. Each slice included a small acceptance checklist (e.g., “expired posts do not appear in /feed”, “link previews fall back gracefully if OG tags are missing”).

Key design choices and alternatives.

Frontend

a. Options considered: Vanilla JS; Angular; React.

b. Decision: React + Tailwind. React’s component model fits card-based feeds; Tailwind accelerates consistent, responsive styles without heavy CSS files.

c. State/data: Lightweight hooks and collocated fetch logic; no heavy global state needed.

d. Trade-offs: React avoids a steep learning curve; Angular provides structure but adds complexity for a student timeline.

Backend & API

a. Options considered: Django/DRF; Laravel; Node/Express.

b. Decision: Node/Express for a unified JavaScript stack, simple middleware, and quick iteration.

c. Security: JWT sessions, input validation, and per-route rate limits; strict CORS using the Vercel origin.

Database

a. Options considered: PostgreSQL; MongoDB.

b. Decision: MongoDB fits document-shaped data (posts, messages, fact-checks) and eases rapid schema evolution; indexes added for feed queries and recency filters.

Messaging Encryption

a. Alternatives: Plaintext at rest; app-level symmetric encryption; full E2EE.

b. Decision: App-level AES-GCM with a base64 32-byte MESSAGE_ENC_KEY. Chosen for clarity and deployability; keys supplied via env; IV/auth tag stored per message.

c. Trade-off: Simpler than full E2EE key exchange; acceptable for a student project with clear documentation.

Link Previews (Unfurl)

a. Alternatives: Client-side scraping; server-side unfurl.

b. Decision: Server-side (/api/links/unfurl) to avoid CORS and expose a consistent, cached preview object. Timeouts and sanitization mitigate slow or malformed sources.

Fact-check & Trust

a. Alternatives: Handwritten rules; community voting; external model.

b. Decision: External model with strict budgets and spacing. Results are optional UI context; authors accumulate a Trust Badge tier via

snapshots.

c. Trade-off: Avoids heavy moderation while giving readers fast, interpretable signals.

Media Storage

a. Alternatives: Local disk; self-hosted object storage; Cloudinary.

b. Decision: Cloudinary (optional) via env toggle, providing CDN delivery and on-the-fly transformations for images.

Deployment

a. Frontend: Vercel, configured with REACT_APP_API_URL=https://<render-app> (no trailing slash).

b. Backend: Render, environment keys for Mongo, JWT, message key, Cloudinary, and trust budgets.

c. Verification: After deploy, confirm /api/links/unfurl returns data; check that pills appear only when results exist.

Risk & Mitigation

a. Rate/Cost spikes => hourly budgets and minimum call spacing.

b. Env misconfigurations => .env.example and startup validation.

c. Legacy localhost URLs => single API client file and CI lint to detect hard-coded hosts.

3.3 Project Plan

The work began with requirements and architecture. I wrote the SRS, mapped the REST API routes, defined the initial MongoDB data models, and prepared the environment variable list for both development and production. This upfront clarity-names, payloads, and secrets—kept later changes small and predictable.

Next came the core experience: the Home feed with time-to-live (TTL) retention. I implemented the Post model, added expiry filters so stale items fall out naturally, and returned a recency-aware feed that proves the “finite scroll” idea in practice. With the feed in place, I introduced Groups and Pages so users could create or join spaces, view content with scoped visibility, and apply delete rules that respect ownership and page/group admin roles.

Private messaging followed. I added AES-GCM encryption at rest with per-message IV and authentication tag, wired the encrypt/decrypt path on the server, and built a small chat UI to test the flow end to end. After that, I implemented server-side link previews. The backend exposes /api/links/unfurl, fetches and sanitizes Open Graph data, and the client renders it in a LinkPreviewCard with graceful fallbacks for minimal pages.

With the basics solid, I layered in credibility features. The fact-check pipeline accepts triggers (engagement thresholds and an optional tag), stores the model result in a compact JSON shape, and surfaces it as an on-post verdict pill. In parallel, an author-level Trust Badge aggregates recent outcomes so readers get a quick, interpretable signal about reliability.

To round out usability, I added search across users and posts and a notifications list with lightweight toasts, keeping the experience responsive without becoming noisy. Governance was then surfaced in the UI and enforced in the API: owners, global admins, and page/group admins can delete within their scopes, and I captured brief audit notes to aid future review.

Before shipping, I spent a pass on hardening: reran functional scenarios end to end, tightened error handling and simple "error budgets" around external calls, removed dead code, and added startup validation for required env keys. Finally, I deployed the backend to Render and the frontend to Vercel, verified CORS and cookie behavior, tested /api/links/unfurl on the hosted domains, committed a clean .env.example, and wrote a short user guide with screenshots. The result is a live, documented prototype that's faithful to the design goals and ready for iterative improvement.

3.4 Task Allocation

Tasks	Weeks																		
	12	14	16	18	20	22	24	26	28	30	32	34	36	38	40	42	44	46	48
Requirements & Architecture																			
Core Platform (Auth, Profiles, Feed + TTL																			
Feature Extensions (Groups/Pages, DMs AES-GCM, Link Unfurl)																			
Trust & Fact-check (triggers, worker, Trust Badge)																			
QA & Deployment (testing, Render+Vercel)																			
Documentation, Ethical Analysis, and Final Report Compilation																			

Estimated	
Actual	

3.5 Summary

This chapter documented how Mask App moved from research and requirements to a concrete, testable design. We prioritized a finite feed and interpretable trust signal over heavy moderation or complex monetization, then selected technologies that a student team can realistically build and operate: React + Tailwind on the client, Node/Express + MongoDB on the server, optional Cloudinary for media, and

environment-based keys for security. The fact-check pipeline is intentionally rate-limited; when results exist, they appear as compact context on the post, while the Trust Badge summarizes an author's history. Messaging at rest is protected via AES-GCM, and deletion rights are scoped (owner/global admin/page/group) to keep governance clear. The plan, roles, and risk controls position the project to evolve-adding diagrams (3.1.2, 3.1.4), expanding tests, and refining budgets-without losing the core objective: reduce time-waste and improve clarity in everyday social feeds.

Chapter 4

Implementation and Results

This chapter explains how Mask App was implemented and evaluated. It documents the runtime environment, the testing approach and cases, key findings from functional and non-functional checks, and a brief discussion of how the implementation addresses the gaps identified earlier.

4.1 Environment Setup

Mask App is implemented on the MERN stack to keep development unified in JavaScript and to simplify deployment.

Frontend: React 18 with Tailwind CSS (component-first styling), Lucid React icons, and small utility hooks for data fetching. The UI lives in src/new-ui/ with feature-focused components such as PostCard, LinkPreviewCard, Trust Badge, and admin panels.

Backend: Node.js (v20) with Express. Routes are organized under /routes (e.g., posts.js, links.js, messages.js, groups.js, pages.js, trust.js, admin routes) with models under /models. Middleware covers JSON parsing, CORS (allow-list), auth (JWT), and static delivery for uploads when enabled.

Database: MongoDB Atlas. Collections store users, posts, groups/pages, messages, notifications, fact-check results, and trust snapshots. Common fields are indexed by recency and author to keep feed and lookups responsive.

Messaging security: Server-side AES-GCM using a base64 32-byte MESSAGE_ENC_KEY; IV and auth tag recorded per message.

Link previews: Server-side unfurl at /api/links/unfurl (time-boxed HTTP fetch, basic sanitization). The client renders results in LinkPreviewCard.

Fact-check & trust: A backend worker calls an external model under rate budgets and minimum spacing. Results are stored as {verdict, explanation, confidence} and aggregated into author trust snapshots read by Trust Badge.

Media: Optional Cloudinary integration (env-controlled) for image storage and transformations.

Configuration:

a. Backend: MONGO_URI, JWT_SECRET, MESSAGE_ENC_KEY (base64-32B), optional CLOUDINARY_URL, CORS_ORIGINS, and trust/fact-check budgets (e.g., hourly caps).

b. Frontend: REACT_APP_API_URL=https://<render-backend> (no trailing slash).

Deployment: Frontend on Vercel; backend on Render. Verification after deploy includes (i) CORS passes for the Vercel origin, (ii) /api/links/unfurl returns OG data, and (iii) pills appear only when a result exists.

Test hosts & tools: Windows 11, Chrome and Edge (latest), mobile viewports via DevTools, Postman for endpoint checks, MongoDB Atlas UI for data validation

4.2 Testing and Evaluation

ID	Feature	Scenario	Input / Setup	Expected	Actual	Status
TC_01	Registration	Create account	Email + password	Account created; JWT issued	Works; redirects to Home	Passed
TC_02	Login	Sign in with valid creds	Registered email + password	Authenticated session	Works; session cookie set	Passed
TC_03	Post create + preview	Post text containing a URL	"Check this: https://example.com"	Server unfurls; preview renders	Preview renders in PostCard	Passed
TC_04	Unfurl fallback	URL without OG tags	Minimal HTML page	Graceful text-only preview	Fallback text shown; no crash	Passed

TC_05	Retention/TTL	Hide stale posts	Seed post older than TTL	Post excluded from feed	Excluded from /feed query	Passed
TC_06	Reactions & comments	Interact on post	React + comment	Counters update; UI refresh	Works as expected	Passed
TC_07	Pages posting	Page admin posting	Admin vs follower	Only page admins can post	Enforced by the backend	Passed
TC_08	Scoped delete	Owner/admin/page or group admin	Try to delete from each role	Authorized deletes succeed	Backend OK; UI button missing for page/group admins	Passed
TC_9	DM encryption	Store ciphertext, verify decrypt	Send message; inspect DB	Ciphertext at rest; valid GCM tag	Ciphertext stored; decrypt on read	Passed
TC_10	Fact-check (tag)	Immediate check on create (demo)	Post with #verify when enabled	Pill with verdict+confidence	Pill appears on applicable posts	Passed
TC_11	Fact-check (engagement)	Auto-trigger by reactions	Hit threshold from unique users	One check stored; pill visible	Single result stored;	Passed

					pill visible	
TC_12	Rate safety	Respect the hourly budget	Exceed the budget in test	No new calls; no pill if skipped	Skipped; no pill shown	Passed
TC_13	Trust Badge update	After a verdict	Author gets result	Badge tier updates	Badge reflects new tier	Passed
TC_14	Notifications	Reaction/mention events	Trigger interactions	Timely notifications	Delivered & visible	Passed
TC_15	Search	Keyword search	Query by username/content	Relevant results	Returns expected items	Passed
TC_16	CORS & base URL	Frontend => Backend	Vercel => Render domain	Requests succeed; cookies sent	Network OK; cookies present	Passed
TC_17	Cloudinary toggle	Upload on/off	Toggle env	Upload routes adapt	Works with/without Cloudinary	Passed
TC_18	Responsive UI	Small screens	360–414 px widths	Readable feed, no overflow	Layout holds; touch targets OK	Passed

4.3 Results and Discussion

The implementation delivers a working prototype that is faithful to the design goals:

Finite, recency-aware feed: The TTL filter keeps stale items out of the Home feed. In use, this creates natural stopping points and prevents long resurfacing chains.

On-post credibility context: When a claim is evaluated, a compact pill shows the verdict and confidence. Because checks are rate-limited and each post is evaluated at most once, the UI avoids noisy or contradictory labels.

Author-level trust: The Trust Badge summarizes recent outcomes so readers can form quick expectations of an author's reliability without leaving the app.

Scoped governance: Owner, global admin, and page/group admins can delete where appropriate; the only remaining work is to expose those controls consistently in the client (TC_09).

Privacy in DMs: Messages are stored as ciphertext (AES-GCM) with per-message IVs. This protects against casual DB inspection while keeping the implementation viable for a student project.

Where it differs from large platforms. The system deliberately excludes heavy monetization stacks, studio-grade editors, and rights-management (e.g., Content ID). Those choices keep complexity and cost down while focusing on the two levers most relevant to attention and comprehension: time (expiry) and trust (on-post context + author tier).

Known limitations and next steps.

1. Add the missing delete control in the UI for page/group admins (backend already supports it).
2. Remove or merge the legacy src/utils/api.js to avoid accidental calls to localhost in production.
3. Expand multilingual support for fact-checks and improve user-visible explanations.
4. Add lightweight analytics panels (non-PII) so admins can review fact-check coverage and budgets.
5. Consider end-to-end messaging in a future phase if the key-management burden is acceptable.

4.4 Summary

This chapter documented Mask App's implementation and the results of testing. The system runs on a pragmatic MERN stack with a React/Tailwind client, an Express API, MongoDB Atlas storage, optional Cloudinary media, and environment-based secrets. Functional tests covered registration, posting, TTL exclusion, link previews, scoped visibility, messaging encryption, fact-check triggers, rate safety, trust aggregation, notifications, search, deployment wiring, and responsiveness. With the exception of a small UI gap on admin delete controls, all critical paths passed. The prototype effectively addresses the key gaps identified earlier by making feeds finite and credibility signals visible, while keeping the application simple enough to operate on student-friendly hosting.

Chapter 5

Engineering Standards and Design Challenges

This chapter outlines the standards and design decisions that shaped Mask App. It covers software, hardware, and communication standards (with alternates and rationale), the project's impact on society, environment, and sustainability, a lightweight financial analysis, and mappings to complex engineering problem categories, knowledge profiles, and engineering activities.

5.1 Compliance with the Standards

Standards help keep the system maintainable, interoperable, and safe. Below we list only what's relevant to Mask App and-where useful-note alternatives with pros/cons and our rationale.

Software Standards

Mask App follows a modular MERN architecture that separates concerns across a React front end, a Node/Express API, and MongoDB for persistence. This structure aligns with common software engineering standards for maintainability and testability while allowing the entire stack to be implemented in JavaScript, which reduces cognitive overhead for a small student team. Alternatives such as Django/DRF or Laravel were considered; both offer strong ecosystems and opinionated patterns, but they introduce a second language and heavier scaffolding than needed for this project's pace. Service boundaries are expressed through a RESTful interface using JSON payloads and predictable resource routes; GraphQL was evaluated for its schema-driven flexibility, and gRPC for its efficiency in service-to-service calls, yet REST remained the pragmatic choice for a browser-centric client and simple caching. Authentication follows established practice with stateless JWT sessions and password hashing using a modern KDF (e.g., bcrypt/argon), whereas server-side sessions were set aside to avoid shared state across deployments, and third-party OAuth was deferred to keep the sign-in surface minimal. For private messaging, at-rest encryption uses AES-GCM with a base64 32-byte key supplied via environment variables; each message stores a unique IV and authentication tag. Full end-to-end encryption was explored but rejected for this phase due to the additional key-exchange UX and recovery complexity. Input validation and rate safety are handled at the middleware layer to enforce request size, shape, and frequency; more elaborate gateway solutions (e.g., Kong/Apigee) were deemed unnecessary at the current scale. Testing emphasizes scenario checks and route probes using lightweight

scripts and Postman collections, with formatting and lint rules to keep the codebase consistent; a full CI matrix is left as future work once the feature set stabilizes.

Hardware Standards

The system targets managed cloud environments to inherit baseline security and availability standards without bespoke server administration. The API runs on Render, the client on Vercel, the database on MongoDB Atlas, and media (when enabled) on Cloudinary's CDN; each provider supplies TLS termination, modern cipher suites, and routine platform patching that would otherwise fall on the developer. A self-hosted alternative-Linux VMs with Nginx/PM2 and an on-prem database—was analyzed but not chosen because it increases the burden of OS hardening, patch cadence, and monitoring for a student project. Within the chosen setup, only the minimum ports are exposed, HTTP is redirected to HTTPS, and write permissions are restricted to controlled upload paths when local storage is enabled. This approach aligns the project with practical hardware/hosting standards while keeping operational risk and cost low.

Communication Standards

All client–server interactions occur over HTTPS (TLS 1.2+), using HTTP/1.1 semantics and JSON as the wire format to maximize browser compatibility and observability. Cross-origin requests are constrained via a CORS allow-list that includes the Vercel origin used by the deployed frontend; this prevents credentialed requests from untrusted sites. Identity is conveyed through JWTs (RFC 7519) signed server-side, delivered as an authorization header or a secure cookie depending on the route. Alternatives such as PASETO or opaque server-stored tokens were evaluated; both are viable, but JWT's ubiquity and tooling fit our deployment pipeline. Where email/OTP flows are used, SMTP over TLS is required and the one-time codes carry short time-to-live values with replay protection. More complex protocol choices such as SOAP/XML or HTTP/2 server push provide no clear benefit for this application and were therefore excluded. Overall, the communication layer follows web standards that are well supported across browsers and hosting providers, minimizing surprises during deployment and maintenance.

5.2 Impact on Society, Environment and Sustainability

This section discusses how Mask App's design choices-finite feeds, lightweight credibility signal, pseudonymous participation, and private messaging-affect individual users, the broader community, and the environment. It also explains the ethical stance that informs moderation and data practices, and outlines a practical plan to keep the system sustainable over time.

5.2.1 Impact on Life

For everyday users, the most immediate effect is a reduction in unproductive screen time. By applying post retention (TTL), the home feed becomes finite instead of endless, which creates natural stopping points without restricting expression. The on-post verdict pill and the author-level Trust Badge give quick, interpretable context so users can judge claims faster and avoid long detours to external sources. Pseudonymous profiles reduce social pressure around participation, which can encourage contributions from people who might otherwise remain silent. At the same time, direct messages are stored as AES-GCM ciphertext, protecting private conversations from casual inspection. A potential risk is over-reliance on machine labels; Mask mitigates this by keeping signal advisory (not hard takedowns), showing them only when a result exists, documenting the meaning of each verdict, and allowing authors to improve their standing over time through better posts.

5.2.2 Impact on Society & Environment

At the community level, small, consistent signals can elevate the quality of discourse: questionable claims carry context, authors build reputations through their histories, and admins have scoped authority (owner, global admin, or page/group admin) to keep spaces healthy. These measures aim to discourage casual misinformation without suppressing legitimate debate. Environmentally, the platform favors managed cloud services and CDN delivery, which are typically more energy-efficient than self-hosted, low-utilization servers. Retention rules reduce long-term storage and the need to reindex or repeatedly serve very old content, trimming compute and network usage over time. If adoption grows, the same design-modular services, caches, and rate-limited external calls-helps keep resource use predictable and allows the team to scale only what is necessary.

5.2.3 Ethical Aspects

Mask App was built with strict ethical considerations at its core. User data is protected through JWT-based authentication and bcrypt password hashing, while private messages are stored with AES-GCM encryption to prevent unauthorized access. Fairness is preserved by applying the same rules to everyone-posts expire by the same TTL logic, fact-checks are advisory rather than punitive, and delete rights are consistently scoped to owners, admins, or group/page admins. The goal throughout has been to respect privacy, ensure clarity, and treat all users equally in how they create, share, and manage content.

5.2.4 Sustainability Plan

Sustainability is addressed on technical, operational, and financial fronts. Technically, the MERN stack and modular services (link unfurl, trust, messaging)

make the system easy to maintain and extend; environment-based configuration allows the same code to run locally and in the cloud. Operationally, rate budgets and minimum intervals for the fact-check worker cap costs and reduce load spikes; a .env.example, startup validation, and lightweight runbooks help future contributors deploy safely. Financially, the platform is designed to operate on free or low-cost tiers (Vercel, Render, MongoDB Atlas, optional Cloudinary) with predictable spending tied to explicit budgets for external model calls. The roadmap favors incremental upgrades, multilingual checks, clearer explanations, optional end-to-end messaging—so improvements can be added without rewriting the foundation or increasing environmental footprint disproportionally to the benefit.

5.3 Project Management and Financial Analysis

Mask App intentionally uses free-tier or low-cost services. Numbers below are estimates (subject to provider pricing) to show feasibility and alternatives.

Table 5.1: Project Management and Financial Analysis

Category	Item	Est. Cost (BDT/month)	Notes
Hosting	Vercel (frontend)	0	Hobby/free tier.
API	Render (backend)	0	Free tier (OK if sleeping).
Database	MongoDB Atlas M0	0	Free tier.
Media	Cloudinary	0	Free tier (transform limits).
Domain	.com/.site	83	≈ 1,000 BDT/year ÷ 12.
Fact-check model	External API	200	Keep low via hourly budgets/intervals (0–300 typical).
Internet	Home broadband	900	Typical student plan (≈ 700–1,200).
Electricity share	Laptop + router	200	Your portion in a shared flat/dorm.
Misc/printing	Handouts, stationery	100	Occasional prints/photocopies.
Total (approx.)		**1,633**	Includes a 10% contingency on the subtotal (~150 BDT).

5.4 Complex Engineering Problem

Building Mask App surfaced complex engineering issues beyond ordinary coding. Designing a finite feed with auto-expiring posts required precise TTL logic and fair

rules, while fact-checks had to balance accuracy, latency, and cost by keeping results advisory and rate-limited. Messaging security brought applied cryptography through AES-GCM with environment-supplied keys, and deployment tested my care with CORS, cookies, and environment variables across Vercel and Render. Even moderation demanded consistent enforcement of scoped delete rules in both backend routes and the UI. These challenges spanned usability, security, reliability, and ethics, and solving them meant applying specialist knowledge, iterative testing, and careful trade-offs to keep the app practical and trustworthy.

5.4.1 Complex Problem Solving

In this section, provide a mapping with problem solving categories. For each mapping add subsections to put rationale (Use Table 5.1). For P1, you need to put another mapping with Knowledge profile and the rational thereof.

Table 5.2: Mapping with Complex Engineering Problem.

EP1 - Depth of Knowledge	EP2 - Range of Conflicting Requirements	EP3 - Depth of Analysis	EP4 - Familiarity of Issues	EP5 -Extent of Applicable Codes/Standards	EP6 - Stakeholder Involvement	EP7 - Interdependence
✓	✓	✓	✓	✓	✓	✓

EP1: Building Mask App meant applying the full MERN stack in practice-React on the client, Node/Express for the backend, and MongoDB for storage. On top of that, I had to work with cryptography for AES-GCM message encryption and integrate an external API for fact-checking, which pushed my knowledge far beyond just standard web coding.

EP2: Many requirements pulled in opposite directions. Fact-checks needed to be accurate but also fast and cheap; users wanted pseudonymity, but abuse had to be prevented; credibility signals needed to be shown clearly but without overwhelming or discouraging users. Balancing these was a constant trade-off.

EP3: Careful analysis went into designing TTL rules for posts so the feed stayed fresh, setting thresholds that decide when a post should be fact-checked, and working out how verdicts feed into a Trust Badge that people can actually understand. Each step required testing different scenarios and adjusting until it felt right.

EP4: Using the MERN stack and REST APIs was familiar territory, but handling encryption details, budgeting external API calls, and solving tricky CORS errors were areas I had to research and learn as I went. These less-documented parts took the most problem-solving effort.

EP5: The project relied on standard web practices rather than anything proprietary. I used HTTPS/TLS for secure connections, JWT (RFC 7519) for sessions, JSON over HTTP for APIs, and followed secure cookie handling. These standards helped keep the system safe and reliable.

EP6: The app wasn't just about coding-it had to meet the needs of users, admins, and moderators, while also staying within the limits of the external fact-check provider's policies and quotas. Each group influenced how features were designed and implemented.

EP7: Different parts of the system were closely linked. The TTL feed, the fact-check worker, trust badge updates, UI labels, and moderation logic all had to line up. If one part broke or changed, the others had to be checked too, so integration and testing were critical.

Mapping with Knowledge Profile

This section is designed to map the overall problem to the Knowledge Profile.

Table 5.3: Mapping with knowledge Profile.

K1 Natural Science	K2 Mathematics	K3 Eng. Fundamentals	K4 Specialist Knowledge	K5 Eng. Design	K6 Eng. Practice	K7 Comprehension	K8 Research Literature
	✓	✓	✓	✓	✓	✓	✓

K2: The project needed some basic probability and math thinking, especially for setting thresholds in the trust system. For example, deciding when a post should trigger a fact-check meant working with reaction counts, spacing between checks, and small rate models. It wasn't heavy math, but it had to be practical enough to keep the system fair and cheap.

K3: Full-stack development skills were applied across the whole build-React for the frontend, Node/Express for the backend, and MongoDB for storing posts and messages. Networking and web security concepts came in too, since the client, API, and database all had to talk smoothly with each other without leaking data.

K4: Specialist knowledge was required in a few areas. AES-GCM encryption for private messages had to be implemented correctly, including IV and authentication tags, which was not something I used before. Also, integrating the Gemini API for fact-checks brought its own learning curve, from handling quotas to parsing results.

K5: Engineering design principles showed up when shaping how posts expire (TTL logic), how verdict pills display in the feed, and how Trust Badges update. These design decisions needed to balance clarity for users with technical limits, so that credibility signals were helpful but not confusing.

K6: Practical engineering skills like debugging, logging, and basic CI/CD were important to keep the app running. A lot of time went into chasing CORS errors, broken tokens, and deployment issues across Vercel and Render. These hands-on practices turned out to be just as critical as the higher-level design.

K7: Understanding user needs shaped much of the system. Regular users wanted privacy and fairness, admins needed reliable moderation tools, and the platform itself had to keep costs under control. Balancing all of these perspectives guided many of the choices in the app.

K8: Past research and literature around misinformation, feed design, and moderation gave context for many of the features. Looking at how other social apps handled fact-checks, badges, or TTL feeds helped avoid repeating the same mistakes and guided the adoption of approaches that were proven to work elsewhere.

Engineering Activities

In this section, provide a mapping with engineering activities. For each mapping, add subsections to put rationale (Use Table 5.3).

Mapping with Complex Engineering Activities

This section is designed to map the overall problem and EAs.

Table 5.4: Mapping with Complex Engineering Activities.

EA1 - Range of Resources	EA2 - Level of Interaction	EA3 - Innovation	EA4 - Consequences for Society & Environment	EA5 - Familiarity
✓	✓	✓	✓	✓

EA1: The development of Mask App relied on a mix of open-source tools and student-friendly resources to keep costs low but still deliver a full system. The MERN stack (React frontend, Node/Express backend, MongoDB Atlas) gave a complete foundation, and hosting was handled with free tiers on Vercel for the client and Render for the API. For media handling Cloudinary was added, while personal laptops and simple setups were used for testing and debugging. This way the project stayed affordable but still worked like a real-world platform.

EA2: The app demanded constant interaction between its parts. The React frontend had to communicate smoothly with the backend API, which in turn connected with MongoDB for storage. At the same time, the backend worker handled fact-check requests and then pushed results back to the UI as credibility pills and Trust Badges. Direct messaging also depended on encryption and decryption happening on demand. Keeping all of these flows consistent required a lot of design adjustments and repeated testing.

EA3: Mask App introduced several new ideas tailored to its purpose. The finite feed with TTL ensured that posts expired automatically, reducing endless scrolling. On-post fact-check pills and aggregated Trust Badges gave users a quick view of content credibility without hiding posts. Encrypted messaging through AES-GCM was added to protect privacy, while scoped moderation made sure that only owners or admins could remove posts. These innovations aimed to keep the app practical but also different from mainstream networks.

EA4: The project also carries social value. By enforcing post expiry, Mask App discourages unhealthy time-wasting behavior that is common on big platforms. Fact-checking and trust signals give readers a better chance to spot misinformation, which can reduce confusion. From an environmental view, the system runs on lightweight cloud services with modest resource usage, so its footprint is minimal. The platform design also supports inclusivity, letting users join anonymously if they prefer, while still keeping safety checks in place.

EA5: While the core stack (React, Node.js, MongoDB) was familiar to work with, several parts required learning from scratch. Setting up AES-GCM encryption with IV and auth tags was new and tricky, and managing fact-check API quotas meant digging into documentation and experimenting. Handling CORS errors across Vercel and Render was another less-documented area that needed trial and error. Balancing between what I already knew and what I had to learn allowed the system to grow while also improving my own skills as a developer.

5.5 Summary

This chapter documented how Mask App aligns with relevant software and communication standards (REST/JSON, HTTPS/TLS, JWT) while adopting pragmatic choices for security (AES-GCM at rest) and deployability (Render/Vercel, Atlas, Cloudinary). We discussed societal, ethical, and sustainability implications of finite feeds and interpretable trust signals, provided two realistic budgets (near-zero and modest paid tiers), and mapped the work to complex engineering categories, knowledge profiles, and activities. The resulting design keeps the app simple to operate in an academic setting while addressing two hard problems-time and trust-with transparent, user-respecting interventions.

Chapter 6

Conclusion

This chapter closes the report by summarizing the project's contributions, acknowledging current limitations, and outlining concrete directions for future work.

6.1 Summary

Mask App set out to explore a pragmatic alternative to "infinite scroll" social feeds by combining two levers-time and trust-within a familiar social experience. On the time side, automatic post retention (TTL) keeps the feed finite and recency-aware so stale items taper off naturally. On the trust side, a server-side fact-check pipeline attaches a compact verdict pill (verdict + confidence) to qualifying posts, while an author-level Trust Badge aggregates historical outcomes into an interpretable reputation cue. These interventions are intentionally lightweight: content is not removed by default; instead, readers get context and can decide.

Built on a MERN stack, the system includes the core social features-profiles, posts, reactions, comments, shares, bookmarks, groups/pages with scoped visibility, basic search, and notifications-plus AES-GCM–protected direct messages and server-side link previews. Admin controls follow clear scope (owner OR global admin OR page/group admin) to keep governance transparent. The codebase is deployment-ready on student-friendly infrastructure (frontend on Vercel, backend on Render, MongoDB Atlas, optional Cloudinary), with configuration driven by environment variables. Functional tests verified end-to-end behavior for TTL, unfurling, scoped moderation, encryption-at-rest, fact-check triggering, budget enforcement, and Trust Badge updates. Aside from a minor UI gap (exposing delete to page/group admins in the client), the prototype works reliably and demonstrates that finite feeds plus interpretable credibility signals can reduce unproductive time while preserving usability and pseudonymity.

6.2 Limitation

Coverage and explanations for fact-checks: Language/domain coverage is limited; verdicts may occasionally misclassify borderline opinion/satire. Rate budgets and spacing mean some posts are intentionally **not** checked, so no pill appears even when a claim exists.

Messaging security scope: DMs are encrypted at rest (AES-GCM) but are not full end-to-end encrypted; key exchange and recovery UX are out of scope for this phase.

Client gaps and legacy code: The UI currently hides the delete control from page/group admins (backend allows it). A legacy src/utils/api.js with a localhost base URL should be removed or unified. Model/file duplication (e.g., older user/post models) can confuse contributors.

Feature breadth: No stories/AR filters, creator monetization, advanced analytics dashboards, verified identity badges, or rights-management (e.g., Content ID)-deliberate exclusions to keep scope focused and costs low.

Delivery and alerts: Notifications are basic; there is no dedicated mobile app and limited real-time push beyond standard refresh intervals.

Governance depth: Appeals workflows, richer audit logs, and multilingual policy pages are minimal; moderation is intentionally simple.

6.3 Future Work

Grow beyond MVP: Mask App began with finite feeds and credibility signals. The next update is adding depth while keeping those core promises.

Richer media & real-time: Expand more than just text/images into albums, video (short and long), GIFs, voice notes, and light editing. Live features with chat and reactions, plus ephemeral stories and polls, will complement the feed.

Groups, Pages & notifications: Add join approvals, pinned rules, and topic tags for groups. Notifications should move to real-time with WebSockets or SSE, plus pause mode and digest emails.

Trust, safety & governance: Fact-checks should support multiple languages, clearer notes, and periodic rechecks. Moderation needs simple tools-hide, delete, mute-with audit logs and appeals. Spam can be managed by adaptive limits and cooldowns.

Messaging & privacy: Move from encryption at rest to optional end-to-end encryption with per-chat keys and secure attachments. Group chats, mentions, voice notes, and timed messages can follow.

Discovery & search: Build a light recommender blending freshness with trust signals. Search should handle typos, filters, and saved queries.

Accessibility & localization: Support Bangla and English, proper text direction, and WCAG basics like keyboard navigation, screen reader labels, and alt text.

Admin tools & analytics: Simple dashboards for fact-check budgets, trust tiers, and reliability stats. Keep operator views clear with latency/error/capacity indicators.

Performance & security: Use queues for background work, caching for media, and graceful fallback when fact-check budgets run out. Continue strengthening security with secret rotation, 2FA, login alerts, and clear data retention rules.

References

[1] V. Bhatia, T. Jain, H. Choudhary, and N. Chawla, "Scrolling in Silence: Is Social Media Bridging Connections or Fueling Isolation?," *Annals of Indian Psychiatry*, pp. 10–4103, 2025.

[2] N. D. Estolatan-Lama, B. E. Isolana, and O. R. Castañares, Jr., "Social Media: Misinformation and Trustability," unpublished.

[3] M. Rodríguez-Ibánez, A. Casánez-Ventura, F. Castejón-Mateos, and P.-M. Cuenca-Jiménez, "A review on sentiment analysis from social media platforms," *Expert Systems with Applications*, vol. 223, p. 119862, 2023.

[4] H. Taherdoost, "Enhancing social media platforms with machine learning algorithms and neural networks," *Algorithms*, vol. 16, no. 6, p. 271, 2023.

[5] D. V. Gunasekeran *et al.*, "The impact and applications of social media platforms for public health responses before and during the COVID-19 pandemic: systematic literature review," *Journal of Medical Internet Research*, vol. 24, no. 4, p. e33680, 2022.

[6] T. Aichner, M. Grünfelder, O. Maurer, and D. Jegeni, "Twenty-five years of social media: A review of social media applications and definitions from 1994 to 2019," *Cyberpsychology, Behavior, and Social Networking*, vol. 24, no. 4, pp. 215–222, 2021.

[7] M. D. Szeto, A. Mamo, A. Afrin, M. Militello, and C. Barber, "Social media in dermatology and an overview of popular social media platforms," *Current Dermatology Reports*, vol. 10, no. 4, pp. 97–104, 2021.

[8] I. Owuor and H. H. Hochmair, "An overview of social media apps and their potential role in geospatial research," *ISPRS International Journal of Geo-Information*, vol. 9, no. 9, p. 526, 2020.

[9] M. Brough, I. Literat, and A. Ikin, "'Good social media?': Underrepresented youth perspectives on the ethical and equitable design of social media platforms," *Social Media + Society*, vol. 6, no. 2, p. 2056305120928488, 2020.

[10] C. Gerlitz, A. Helmond, F. N. van der Vlist, and E. Weltevrede, "Regramming the platform: Infrastructural relations between apps and social media," *Computational Culture: A Journal of Software Studies*, vol. 7, 2019.

[11] A. Helmond and F. N. van der Vlist, "Social media and platform historiography: Challenges and opportunities," *TMG—Journal for Media History*, vol. 22, no. 1, pp. 6–34, 2019.

[12] M. Naeem, "Uncovering the role of social media and cross-platform applications as tools for knowledge sharing," *VINE Journal of Information and Knowledge Management Systems*, vol. 49, no. 3, pp. 257–276, 2019.

[13] C. Montag, B. Lachmann, M. Herrlich, and K. Zweig, "Addictive features of social media/messenger platforms and freemium games against the background of psychological and economic theories," *International Journal of Environmental Research and Public Health*, vol. 16, no. 14, p. 2612, 2019.

[14] A. A. Alalwan, N. P. Rana, Y. K. Dwivedi, and R. Algharabat, "Social media in marketing: A review and analysis of the existing literature," *Telematics and Informatics*, vol. 34, no. 7, pp. 1177–1190, 2017.

[15] S. Picazo-Vela, I. Gutiérrez-Martínez, and L. F. Luna-Reyes, "Understanding risks, benefits, and strategic alternatives of social media applications in the public sector," *Government Information Quarterly*, vol. 29, no. 4, pp. 504–511, 2012.

[16] S. Vosoughi, D. Roy, and S. Aral, "The spread of true and false news online," *Science*, vol. 359, no. 6380, pp. 1146–1151, 2018.

[17] D. M. J. Lazer *et al.*, "The science of fake news," *Science*, vol. 359, no. 6380, pp. 1094–1096, 2018.

[18] H. Allcott and M. Gentzkow, "Social media and fake news in the 2016 election," *Journal of Economic Perspectives*, vol. 31, no. 2, pp. 211–236, 2017.

[19] K. Shu, A. Sliva, S. Wang, J. Tang, and H. Liu, "Fake news detection on social media: A data mining perspective," *ACM SIGKDD Explorations Newsletter*, vol. 19, no. 1, pp. 22–36, 2017.

[20] C. Wardle and H. Derakhshan, *Information Disorder: Toward an Interdisciplinary Framework for Research and Policy Making*, Council of Europe Report DGI(2017)09, 2017.

[21] M. J. Dworkin, *NIST Special Publication 800-38D: Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC*, National Institute of Standards and Technology, 2007.

[22] W3C, "WebRTC 1.0: Real-Time Communication Between Browsers," W3C Recommendation, Jan. 26, 2021.

[23] I. Fette and A. Melnikov, "The WebSocket Protocol," RFC 6455, IETF, Dec. 2011.

[24] M. Cha, H. Haddadi, F. Benevenuto, and K. P. Gummadi, "Measuring user influence in Twitter: The million follower fallacy," in *Proc. Int. AAAI Conf. Weblogs and Social Media (ICWSM)*, 2010, pp. 10–17.

[25] D. M. Boyd and N. B. Ellison, "Social network sites: Definition, history, and scholarship," *Journal of Computer-Mediated Communication*, vol. 13, no. 1, pp. 210–230, 2007.

213-15-4425

ORIGINALITY REPORT

14%	9%	5%	11%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Submitted to Daffodil International University Student Paper	4%
2	Submitted to Monash University Student Paper	3%
3	Submitted to United International University Student Paper	1%
4	openaccess.altinbas.edu.tr Internet Source	<1%
5	dspace.daffodilvarsity.edu.bd:8080 Internet Source	<1%
6	cahaya-ic.com Internet Source	<1%
7	Submitted to Ashesi University Student Paper	<1%
8	Mohammed Nasser Al-Suqri, Maryam Gillani. "A Comparative Analysis of Information and Artificial Intelligence towards National Security", IEEE Access, 2022 Publication	<1%
9	Areej Babiker, Sameha Alshakhsi, Cornelia Sindermann, Christian Montag, Raian Ali. "Examining the growth in willingness to pay for digital wellbeing services on social media: A comparative analysis", Heliyon, 2024 Publication	<1%
10	Submitted to Gardner-Webb University Student Paper	<1%

23% detected as AI

The percentage indicates the combined amount of likely AI-generated text as well as likely AI-generated text that was also likely AI-paraphrased.

Caution: Review required.

It is essential to understand the limitations of AI detection before making decisions about a student's work. We encourage you to learn more about Turnitin's AI detection capabilities before using the tool.

Detection Groups

53 AI-generated only 23%
Likely AI-generated text from a large-language model.

1 AI-generated text that was AI-paraphrased 1%
Likely AI-generated text that was likely revised using an AI-paraphrase tool or word spinner.

Frequently Asked Questions

How should I interpret Turnitin's AI writing percentage and false positives?
The percentage shown in the AI writing report is the amount of qualifying text within the submission that Turnitin's AI writing detection model determines was either likely AI-generated text from a large-language model or likely AI-generated text that was likely revised using an AI paraphrase tool or word spinner.

False positives (incorrectly flagging human-written text as AI-generated) are a possibility in AI models.

AI detection scores under 20%, which we do not surface in new reports, have a higher likelihood of false positives. To reduce the likelihood of misinterpretation, no score or highlights are attributed and are indicated with an asterisk in the report (*%).

The AI writing percentage should not be the sole basis to determine whether misconduct has occurred. The reviewer/instructor should use the percentage as a means to start a formative conversation with their student and/or use it to examine the submitted assignment in accordance with their school's policies.

What does 'qualifying text' mean?
Our model only processes qualifying text in the form of long-form writing. Long-form writing means individual sentences contained in paragraphs that make up a longer piece of written work, such as an essay, a dissertation, or an article, etc. Qualifying text that has been determined to be likely AI-generated will be highlighted in cyan in the submission, and likely AI-generated and then likely AI-paraphrased will be highlighted purple.

Non-qualifying text, such as bullet points, annotated bibliographies, etc., will not be processed and can create disparity between the submission highlights and the percentage shown.