Warning: the content is WIP and is not representative of final work

VIETNAM NATIONAL UNIVERSITY HO CHI MINH CITY
HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

REPORT:
CAPSTONE PROJECT HK252-DATN-328:
DEVELOP A SOCIAL NEWS WEBSITE ORIENTED TOWARDS COMPUTER SCIENCE FOR HCMUT
SEMESTER 252 ACADEMIC YEAR 2025-2026

BKHack: A Computer Science Social News Website for HCMUT

Major:	**Computer Science**
Council:	Department of CSE
Supervisor(s):	Dr. Trương Tuấn Anh
	MS. Nguyễn Minh Tâm
Reviewer:	MS. Mai Đức Trung

_____o0o_____

Student 1:	**Lê Nguyễn Gia Bảo**	**2210216**
Student 2:	Hồ Gia Tường	2252887
Student 3:	Lê Công Minh Khang	2252295

HO CHI MINH CITY, May 2026

Abstract

We present **BKHack**, a social news website oriented towards computer science, designed for internal use at Bach Khoa University of Ho Chi Minh City. The platform provides a centralized space for students to share knowledge, discuss research topics, and stay up-to-date with developments in the field.

Throughout this report, we mainly discuss the analysis, the design and several implementation details of BKHack. The analysis involves user-level specifications of features and also non-functional requirements that will help us achieve feature goals. The design includes high-level architectural diagrams and other developer-oriented specifications. The "implementation" section involves intiial implementations of ideas presented in the design section. We also provide a timeline which should realistic map our development projection for the next phase of this project.

In conclusion, we've gone through the initial development phase of this project with a working prototype based on this report.

Guanrantee of originality

This capstone project is an original work of our group under the supervision of Mr. Trương Tuấn Anh. All content in this capstone are entirely of our own and free from plagiarism.

The tables and figures in this paper were collected from various sources. These sources are cited in the "References" section or footnoted directly within the page.

Were any form of academic misconduct to be detected, we as the BKHack developers shall take full responsibility. Ho Chi Minh City University of Technology (HCMUT) shall bear no liability for any copyright or intellectual property infringements committed by us during this project.

The BKHack developers

Acknowledgements

We would like to express our gratitude to the Hồ Chí Minh University of Technology (HCMUT) for providing us with opportunities to study, practice, connect; to attain the necessary knowledge and skills to implement this project.

We are indebted to Mr. Trương Tuấn Anh, Mr. Nguyễn Minh Tâm, and other advisors of the Faculty of Computer Science and Engineering for providing feedback and guidance for this software engineering project.

Due to our limited experience and knowledge, some shortcomings are unavoidable. We sincerely look forward to receiving comments and advice from our reviewers.

The BKHack developers

To Tùng, our mutual friend, who originally proposed the concept for this project, a warm and sensible person who gave feedback on ideas, warned and encouraged us on the way.

Tường and Bảo

To Triết, the smart and kind brother I don't deserve, who encouraged me throughout this journey.

Bảo

Contents

Figures

1. Overview

1.1. Context

Within the academic environment of Ho Chi Minh City University of Technology (HCMUT), we as students of the university feel that there is no dedicated platform for academic computer science discourse. Academic exchanges currently lack aggregation - they are dispersed across informal and external channels such as social media groups, private chats, and public forums. A potential effect of this dispersion is that it may fragment information flow, where valuable insights, resources, and discussions are buried under noise or confined to isolated communities. Students and lecturers alike face significant difficulty in procuring, aggregating, and verifying information relevant to their studies or research interests.

Beyond dispersion, another emerging challenge is information reliability over time. Many discussions, particularly those meant to be educational or regarding recent developments, quickly become outdated, disproven, or replaced by better knowledge. As Computer Science majors, we have often searched for technical solutions on discussion and social news platforms on the internet and were often faced with outdated information. On existing platforms, such content persists indefinitely without correction or context. Readers must sift through multiple addenda, clarifications, or contradictory follow-ups across separate threads to reconstruct the "current truth". This is not only a waste of effort but it also erodes academic credibility and engagement.

1.2. Survey

We have done an informal questionaire-based survey in HCMUT, where we ask students and lecturers at the site of HCMUT a few questions about their social news experience in general:

1. "What content are you interested in the most?"
2. "How do you usually discover topics you'd explored in-depth (rabbit holes/deep dives)?"
3. "What makes a piece of news or information credible to you?"
4. "What is the most important aspect of a community platform?"

We found that there's a small albeit relevant interest in an educational, research-focused social news website in HCMUT. There are certain expectations as to how knowledge and news content should be treated. For example, according to Figure 3, citation factors considerably in a content's credibility. These results should influence requirement analysis and design of the system in later sections.

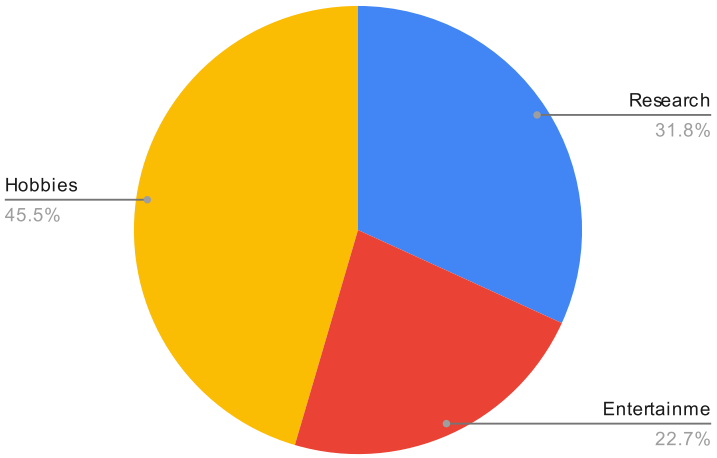

Figure 1: Responses to question 1 of the questoinaire, “what content are you interested in the most?”

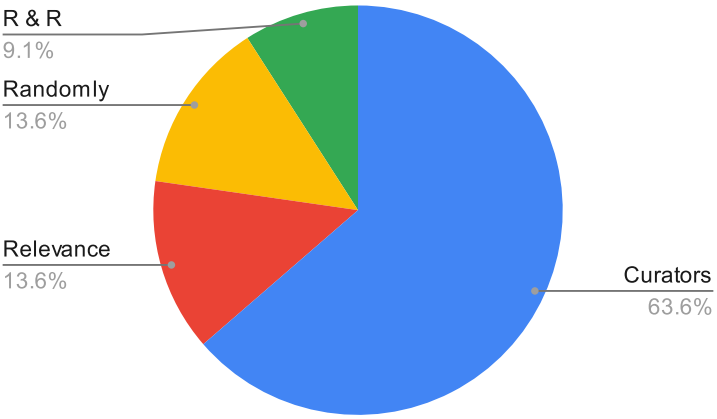

Figure 2: Responses to question 2 of the questoinaire, “how do you usually discover topics you’d explored in-depth (rabbit holes/deep dives)?”

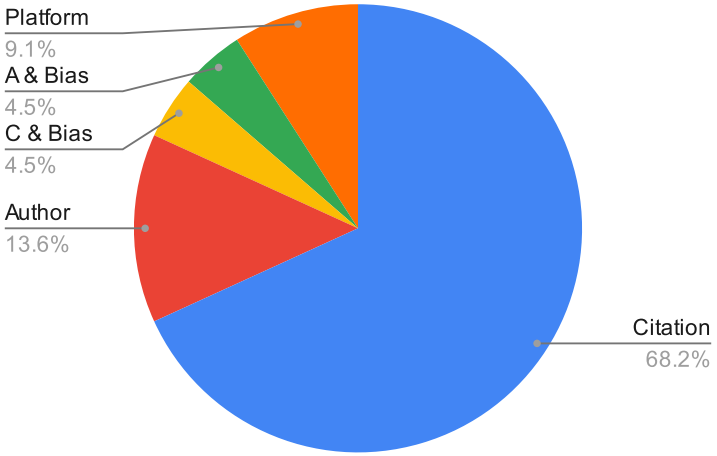

Figure 3: Responses to question 3 of the questionnaire, “what makes a piece of news or information credible to you?”

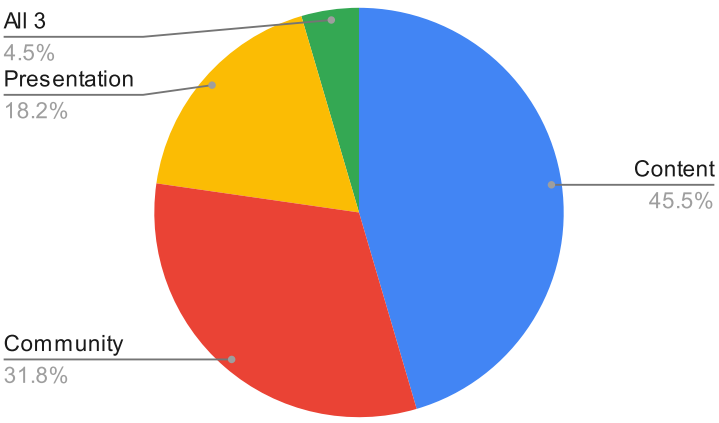

Figure 4: Responses to question 4 of the questoinaire, "what is the most important aspect of a community platform?"

1.3. Solution

In response to these issues, BKHack is conceived as a centralized, university-integrated platform for the procurement, discussion, and curation of computer science knowledge. For the goal of ensuring aggregation and reliability, we propose a novel revision - pull request system for interacting with discussions, inspired by existing systems found in platforms like Wiki.gg and X (formerly Twitter). It combines the community interactivity of social platforms with the structural rigor of version-controlled systems. Through organized threads, revision histories, and contributor accountability, BKHack aims to transform fragmented dialogue into a living, traceable, and evolving body of academic discussion — ensuring that each topic remains accessible, contextualized, and academically sound.

1.4. Goals

The goal of this capstone project is to develop BKHack, a social news website focusing on computer science in HCMUT. This website should achieve the following effects on the community:

- It should provide a trustworthy platform for sharing and discussing news about tech and CS
- It should be an outlet for others to discover and learn about your own personal projects
- It should encourage the pursuit of truth via a community-sourced, fact-checking revision — pull request system.

1.5. Scope

The development of BKHack concerns skills in web application development and the hosting of an always-online internet service.

The stakeholders of BKHack should be all students and lecturers of HCMUT who are interested in computer science discussion. The maintenance of BKHack may also concern system adminstrators of staff of HCMUT.

1.6. Structure

For this semester, we focus on the design of our system, specifications, and other high-level details; these will be the bulk of the content of this report. As for implementation, we only discuss briefly some details, as well as providing a timeline for working in the future.

In Section 1, we provide a high-level overview of this project — its context, goals, and scope.

In Section 3, we analyze related systems to highlight their advantages and disadvantages as foundation, and from there we propose a design for our system BKHack.

In Section 4, we present and discuss our implementation progress of our system BKHack.

In Section 6 we discuss what we've achieved in this project, our shortcomings and future work.

2. Analysis

2.1. Related systems — state of the arts

At Bach Khoa, the LMS system provides a forum discussion feature where, per course, lecturer and course owners can create forums, and students and course participants can create threads and submit posts into these threads whose topics are the scope of the current containing course.

Outside of Bach Khoa, there are social news websites oriented towards computer science, such as Hacker News [1], and Lobsters [2]. However, these do not tailor to the university's specific ecosystem and community conventions.

2.1.1. Hacker News

Figure 5: Hacker News Logo. Source: http://news.ycombinator.com

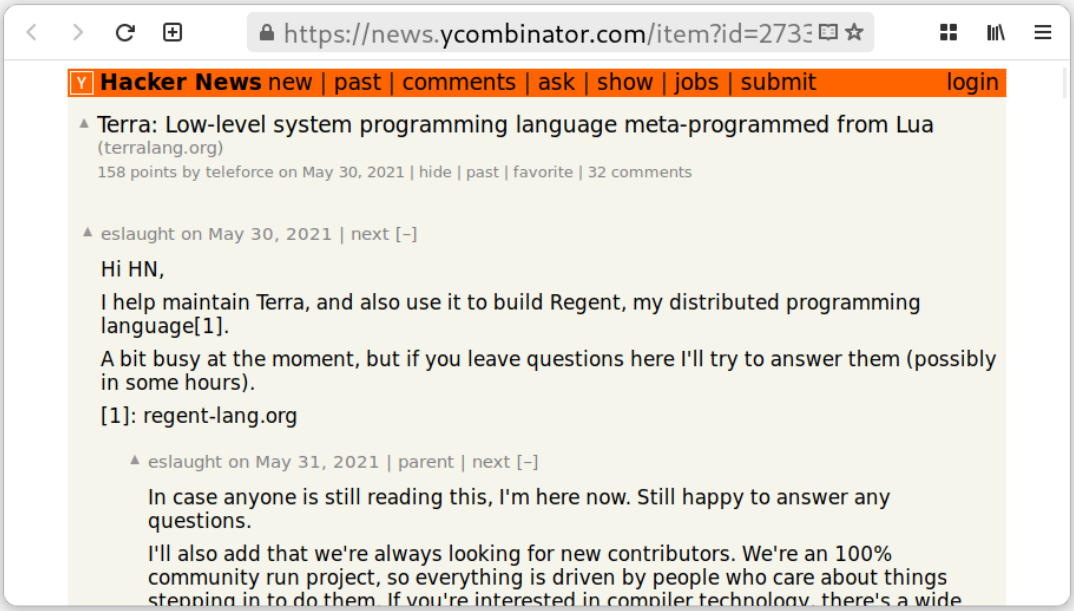

Figure 6: Hacker News Item. Source: window capture at https://news.ycombinator.com/item?id=27334065

Hacker News, operated by Y Combinator, serves as a lightweight, text-oriented community for sharing and discussing technology-related topics [1].

Hacker News's voting-based system naturally promotes popular or timely information, which helps highlight trending developments in computing and research. Yet this same mechanism emphasizes immediacy over persistence: posts lose visibility as new content arrives, and older discussions, even when still relevant, fade from attention. As a result, while the platform excels at capturing current discourse, it offers no structured means to preserve or revisit evolving technical understanding — a limitation when reliability and academic continuity are desired.

2.1.2. Lobsters

Figure 7: Lobsters Logo. Source: https://github.com/lobsters/lobsters/tree/master/app/assets/images

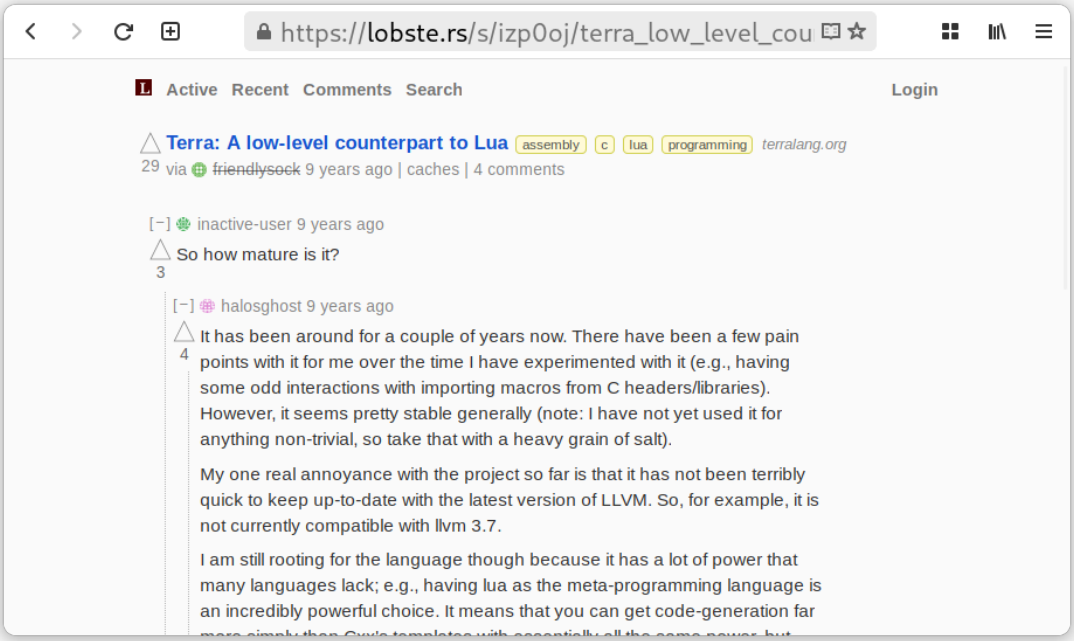

Figure 8: Lobsters Item. Source: window capture at https://lobste.rs/s/izp0oj/terra_low_level_counterpart_lua#c_fthxci

Lobsters, launched in 2012 by Joshua Stein, is a link aggregation and discussion site similar to Hacker News, but with a stronger focus on technical topics and community curation [2]. It features a robust tagging system and supports a smaller, tight-knit community.

Lobsters provides a more tightly moderated environment for technical discussion, emphasizing depth and accuracy within a smaller community. Its tagging system improves organization and helps participants focus on specialized areas of computing. Still, the community's small scale and relative obscurity limit both visibility and diversity of perspectives. Like many social discussion systems, its content stream is chronological and transient, making earlier exchanges difficult to trace once new discussions dominate. Despite stronger curation compared to larger platforms, information there still risks becoming stale or contextually detached as technologies progress.

2.1.3. HCMUT LMS course-based forums

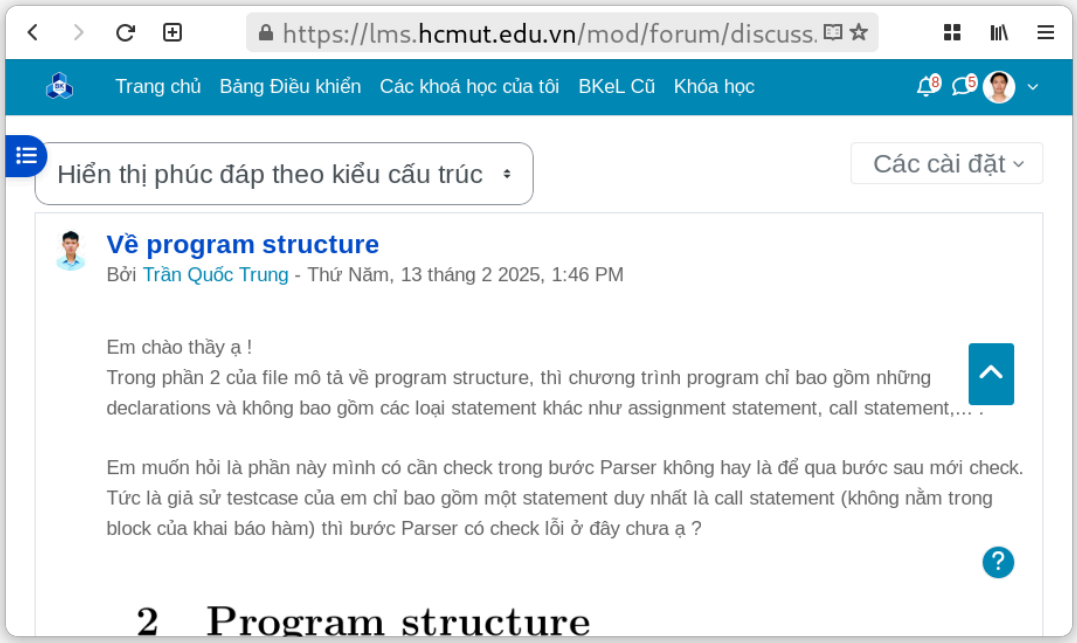

Figure 9: A discussion forum thread on HCMUT LMS. Source: window capture at https://lms.hcmut.edu.vn/mod/forum/view.php?id=500327

The Learning Management System at HCMUT (HCMUT LMS) provides optional course-specific discussion forums where lecturers and students can exchange questions, explanations, and course materials.

HCMUT LMS integration within the academic infrastructure ensures that discussions are relevant to course objectives and directly connected to ongoing instruction. However, such forums may not be provided, and once a course concludes, access to its forum content typically ends, leaving valuable insights inaccessible to future students. Because conversations are confined to individual courses without interconnection or global indexing, information quickly becomes outdated or redundant, and institutional memory of technical discussion is lost over time.

2.1.4. Reddit

Figure 10: Reddit Logo. Source: https://redditbrand.lingoapp.com/a/Reddit-Logo-Wordmark-OrangeRed-Ye1vjW?asset_token=a-o7nPpp4qchR8xJmiWfZh7CaBCdzEjJPGMNUlYE8Dw&v=41

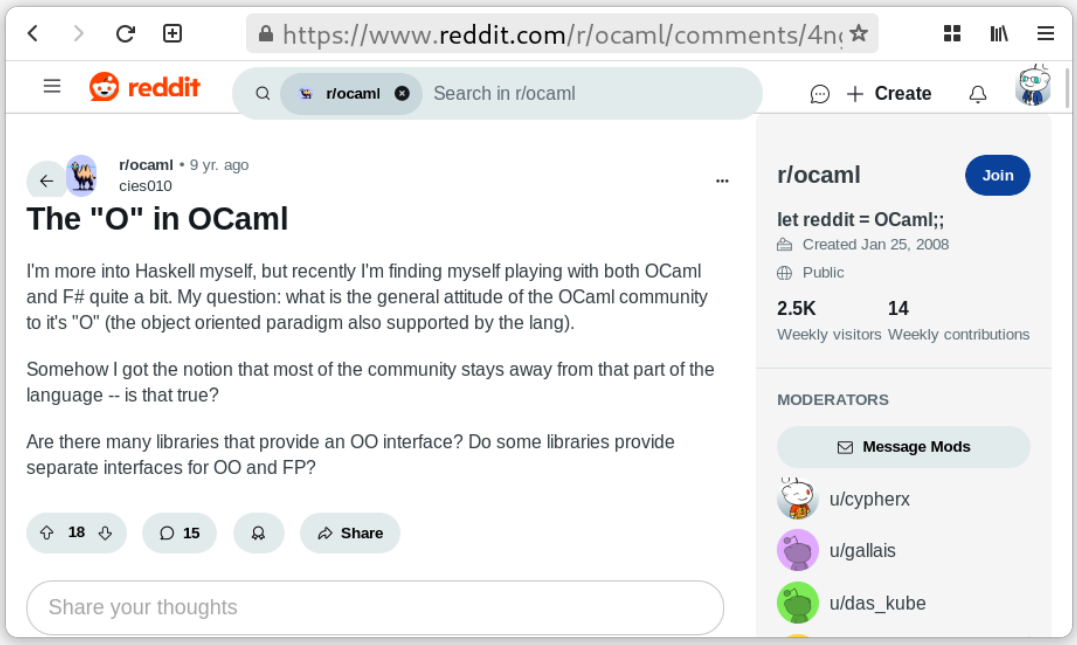

Figure 11: A discussion thread in a Reddit group discussing the programming language OCaml. Source: window capture at https://www.reddit.com/r/ocaml/comments/4ngcb7/the_o_in_ocaml/

Reddit is one of the largest online community platforms, hosting numerous sub-communities (subreddits) across diverse domains [3]. Its familiar interface and interaction mechanisms make it widely accessible and easy to use.

Reddit's subreddit model encourages topic segregation, while familiar interaction patterns make engagement simple. However, the platform prioritizes breadth and user activity rather than long-term knowledge retention. Threads surface and sink according to short-term interest, and technical discussions often become buried under social noise. Since moderation and quality standards vary widely across communities, accuracy and context depend heavily on individual contributors. Information is therefore prone to obsolescence, and the lack of institutional framing limits academic applicability.

2.1.5. Facebook pages and groups

Figure 12: Facebook Logo. Source: https://facebook.com

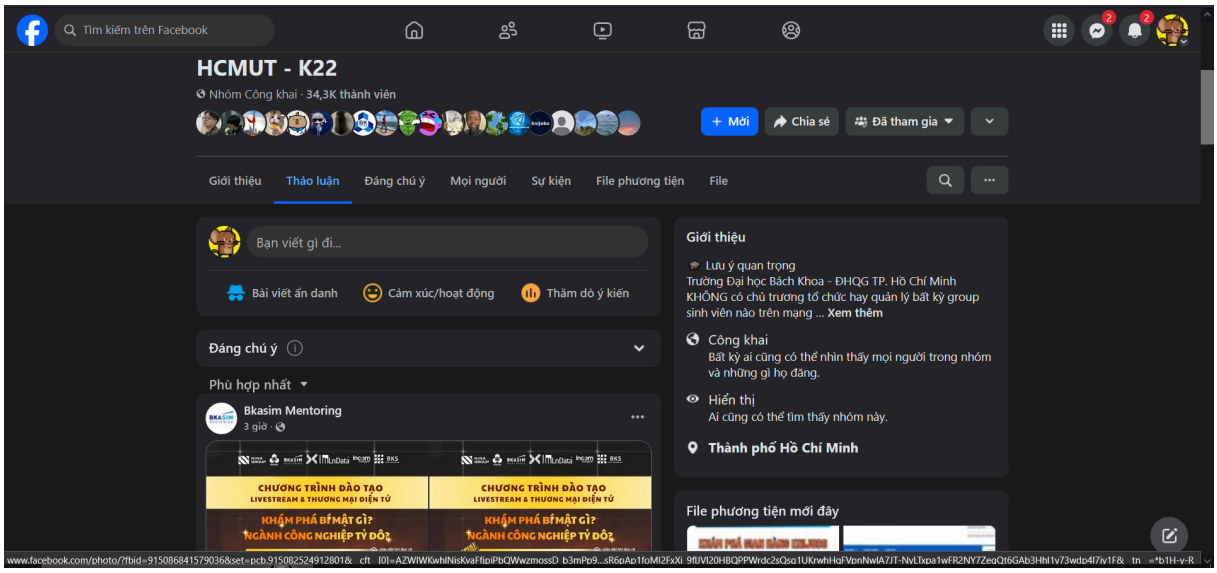

Figure 13: A Facebook community for HCMUT Students. Source: window capture at https://www.facebook.com/groups/hcmut.k22

Among HCMUT students, Facebook groups remain one of the most commonly used tools for coordination, peer support, and information sharing. They benefit from high participation rates and instant notifications, allowing quick dissemination of updates or advice. Yet their structure is entirely social: posts appear chronologically and are rapidly displaced by newer content, leaving older discussions nearly unrecoverable. Search functions are limited, and without formal categorization or validation mechanisms, important information can become outdated or lost in casual conversation. Furthermore, as these groups exist outside university oversight, data persistence and academic reliability cannot be guaranteed.

2.1.6. X (formerly Twitter)

Figure 14: X (formerly Twitter) Logo. Source: http://x.com

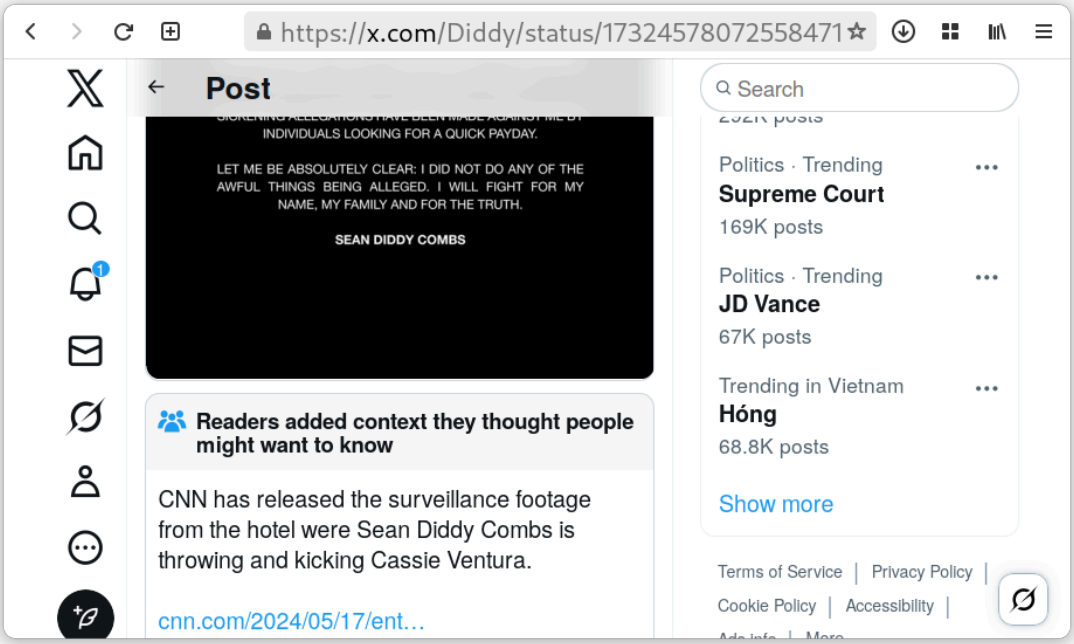

Figure 15: An X (formerly Twitter) page with community note. Source: window capture at https://x.com/Diddy/status/1732457807255847165

X (formerly Twitter) is a social media where users can post their thoughts, media, or news about anything in the world [4]

X's Community Notes feature introduces a collaborative approach to context correction, allowing users to append clarifications or sources to existing posts. This approach effectively mitigates misinformation by letting the community annotate content collectively. However, the added notes remain static once approved; they supplement rather than update the original post. Over time, even corrected content may become misleading as the surrounding context evolves. While the feature demonstrates the value of community-driven accountability, it does not ensure sustained relevance of information — a critical factor in technical and academic discourse.

2.1.7. Wiki.gg

Figure 16: wiki.gg Logo. Source: https://wiki.gg

Figure 17: A community wiki made in wiki.gg. Source: window capture at https://deadcells.wiki.gg/

Wiki.gg is a community-run wikipedia primarily used for maintaining detailed documentation of media from various commuinties [5].

Its structure supports continual content improvement and consistent referencing, helping to preserve the accuracy of technical material. However, its design emphasizes static documentation rather than active discussion. As a result, updates depend on contributor initiative rather than real-time dialogue, and information can lag behind current understanding. Although it succeeds in maintaining structured knowledge, it does not address the social dynamics of discussion or the need to contextualize information as it ages.

2.1.8. Conclusion

All of these platforms act as centers for sharing and discussing information, each with its own strengths and focus areas:

- HCMUT LMS is a system integrated with HCMUT's ecosystem and a university's academic context;
- Hacker News is a technology-focused link aggregation site where users share and discuss articles related to startups, programming, and industry news;
- Lobsters is a similar link-sharing platform that focuses more on the technical side of computing with a detailed tagging system;
- Reddit is a well-known and beginner-friendly platform built around topic-based communities (subreddits), allowing discussion across a wide range of interests;

- Facebook pages and groups are popular across many audiences, Facebook combines social networking with tools for event planning and community building;
- X (formerly Twitter)'s community noting is a community-based feature that helps fact-check posts and add reliable context with linked sources;
- Wiki.gg has editorial features that allow collaborative editing so users can update or correct content as new information becomes available.

Together, these platforms demonstrate different ways of building community-driven spaces for information sharing and discussion. Their individual strengths—such as open conversation, collaboration, and shared moderation will guide our own approach to creating an effective platform to serve the HCMUT students and lecturers.

2.2. Functional requirements analysis

The functional requirements for BKHack are systematically derived from the project's core mission to transition from fragmented, informal academic dialogue toward a centralized, traceable, and evolving body of knowledge.

ID	Description
FR1	The system supports multiple content types, including articles, discussions, issues, pull requests, revision history entries, and notes.
FR2	All content types support creation, viewing, editing, and deletion according to user permissions.
FR3	All content lists—posts, notes, issues, pull requests, revision histories, users, and feeds—support rich filtering, searching, sorting, and collapsing.
FR4	Article posts maintain complete version history, recording all edits, administrative actions, ownership transfers, and reverts.
FR5	Users can view diffs between any two article versions.
FR6	Posts allow inspecting and updating editorial permissions.
FR7	Users may create issues associated with any article post.
FR8	Users may create pull requests proposing changes to any article post.
FR9	Each issue and pull request includes its own discussion thread.
FR10	Pull requests present structured change information, including diffs, comments, and status.

ID	Description
FR11	Authorized users may approve, reject, or modify-and-merge pull requests.
FR12	The system tracks relationships among issues, pull requests, and article versions.
FR13	Articles, issues, and pull requests support threaded discussions.
FR14	Discussion threads inherit global filtering, sorting, and collapsing capabilities.
FR15	Discussions may reference specific versions, sections, or lines of content.
FR16	Users can upvote or downvote comments in discussions, issues, and pull requests.
FR17	All comments are annotated with the article version that existed at the time of posting.
FR18	Users may create lightweight notes, either standalone or referencing other content.
FR19	Notes support creation, editing, searching, sorting, filtering, tagging, and deletion.
FR20	The system provides account creation, authentication, and session management.
FR21	Each user has a profile containing authored content, activity history, contributions, and statistics.
FR22	Users have role-based permissions controlling editing, moderation, and administrative access.
FR23	Users may configure personal preferences, profile details, and notification settings.
FR24	Users may subscribe to content or authors to receive updates.
FR25	The system generates a personalized feed based on subscriptions, activity, and ranking.
FR26	Users can filter and sort feed items using tags, metadata, or relevance criteria.
FR27	Administrators can manage users, posts, tags, and system-wide settings and access analytics.
FR28	Administrators can inspect platform metrics such as growth, activity distribution, and moderation statistics.
FR29	Administrators may review and update global taxonomies, including categories, tags, and classifications.
FR30	Administrators can audit posts, issues, pull requests, and user histories.
FR31	Administrators oversee ownership transfers and moderation outcomes.

2.3. Non-functional requirements analysis

2.3.1. Performance

The system needs to perform well under reasonable load

ID	Description
NFR1	The system should handle at least **30** concurrent users during normal operation without noticeable slowdown.
NFR2	Average server-side response time for common actions (viewing articles, loading discussions, submitting comments) should remain under 400 ms on campus-hosted infrastructure.
NFR3	Full page load time should not exceed **3** seconds on a standard broadband connection.
NFR4	Background processes (indexing, feed generation, notifications, analytics) must not increase foreground request latency by more than 20% under load.

Table 3: NFR for system performance

2.3.2. Reliability and availability

The system must preserve data integrity across edits, revisions, and moderation actions.

ID	Description
NFR5	The system should maintain at least 97% uptime during the evaluation period, excluding scheduled maintenance.
NFR6	Article versions, issues, pull requests, and comments must be durably persisted; no committed revision may be lost after acknowledgment.
NFR7	Automated database backups must occur at least once daily and be retained for 7 days, including revision histories and audit logs.
NFR8	Failure of auxiliary subsystems (analytics, notifications, moderation tools) must not prevent core read/write operations on articles and discussions.

Table 4: NFR for system reliability

2.3.3. Security

The system must protect user accounts, content integrity, and moderation workflows.

ID	Description
NFR9	User credentials must be stored using industry-standard password hashing with salting; sensitive data must never be stored in plain text.
NFR10	The system must be demonstrably resistant to common web attacks through manual or scripted testing, including:
NFR10.1	SQL Injection
NFR10.2	Cross-Site Scripting (XSS)
NFR10.3	Cross-Site Request Forgery (CSRF)
NFR11	Authentication sessions must expire after 20 minutes of inactivity and require re-authentication for sensitive actions (merging, moderation, administration).
NFR12	All data transmission must use HTTPS in deployment environments that support it.

Table 5: NFR for system security

2.3.4. Usability and accessibility

The interface should support complex interactions without overwhelming users.

ID	Description
NFR13	The user interface must remain usable and fully functional on both desktop and mobile screen sizes.
NFR14	Core actions (reading articles, navigating versions, participating in discussions, creating issues or pull requests) should be discoverable within three interactions from primary entry points.
NFR15	The system must provide clear and immediate feedback for all user actions, including edits, submissions, approvals, rejections, and errors.
NFR16	Text layout and color contrast must meet basic accessibility guidelines to ensure readability of long-form and technical content.

Table 6: NFR for system usability

2.3.5. Maintainability and extensibility

The system should accommodate evolving academic and collaborative workflows.

ID	Description
NFR17	The codebase must be structured into modular components reflecting major concerns (content, revisions, discussions, authentication, moderation, feeds).
NFR18	Each major module must be independently testable with clear interfaces and minimal coupling.
NFR19	At least 60% automated test coverage is expected for core business logic, including revision tracking and permission enforcement.
NFR20	The architecture must support future extension of content types, workflows, or ranking algorithms without requiring fundamental redesign.

Table 7: NFR for system maintainability

2.3.6. Documentation and developer experience

The project should be understandable and sustainable for future contributors.

ID	Description
NFR21	The project must include comprehensive developer documentation, including:
NFR21.1	A setup guide for local development and deployment
NFR21.2	Documentation of core modules, data models, and APIs
NFR21.3	Inline comments for non-trivial logic, particularly revision handling and permission checks
NFR24	Version control practices must follow a defined workflow, including meaningful commits and review where applicable.

Table 8: NFR for developer experience

2.3.7. Ethical and community standards

The platform must support transparent, respectful academic collaboration.

ID	Description
NFR25	A clear code of conduct and moderation policy must be publicly accessible to all users.
NFR26	User data handling must comply with university privacy policies and ethical research guidelines.
NFR27	Testing and development activities must not expose or rely on real student data without explicit consent.

Table 9: NFR for ethics

3. Design

3.1. Design goals

In implementing BKHack, it's useful to have design goals in mind. These design goals are mostly derived from our high-level project-wide goals, as discussed in Section 1.4, to benefit them as best as possible. They also reflect the personal opinions of our development team and can be used to generally smoothen our decision-making process.

We have kept these constant considerations in mind:

1. Using UX ergonomics, how to direct users to interact with integrity?
2. How to ensure great maintainability, and reconcile them with performance?
3. In architecture, be direct; or, how to minimize the amount of layering in communication?

For (1). Some examples. Page for discussion takes place after page for viewing article, ensuring that the readers must have read the article first before making decisions.

For (2), we achive by ensuring maximum reasonability and expressiveness in development (e.g. in source code). For example, most diagrams in documentation are coded in domain-specific languages. We discuss more in Section 4.1.3.

We achieve by various means. We use language-integrated query (LINQ) [6] to embed database queries in frontend – specifically, we used QueL, a SQL eDSL for OCaml using the tagless-final style of embedding [7].

For (3), we adopt a service-oriented architecture (SOA) [8] for our system architecture where our data layer is provided by the Firebase DaaS with several built-in services for authentication.

3.2. Architectural views

In documenting architectures for the BKHack system, we follow the Views and Beyond method [9]. By adopting its central advice [10] , we've selected and drawn the diagrams in this sections which illustrate the most relevant and important characteristics of our system.

3.3. Use cases

3.3.1. Actors

Actors that interacts with the system: User: Signed-in students and lecturers Admin: Users with moderation or administrative privileges

3.3.2. Usecase diagram

3.3.2.1. Authentication & accounts

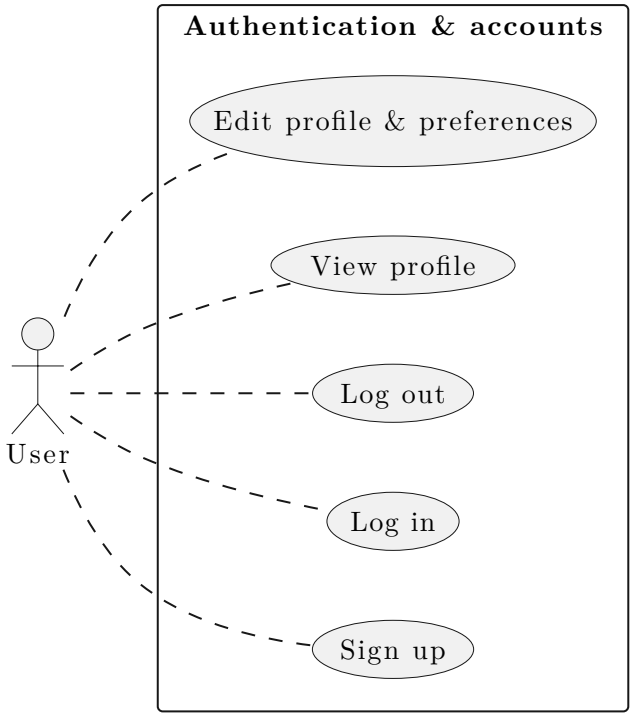

Figure 18: The use case diagram of authentication & accounts

Usecase ID	Name	Primary Actor	Goal
UC01	Sign up	User	Create a new user account
UC02	Log in	User	Authenticate using credentials or identity provider
UC03	Log out	User	Terminate authenticated session
UC04	View profile	User	View a user profile and activity history
UC05	Edit profile & preferences	User	Update profile details and notification settings

3.3.2.2. Content consumption

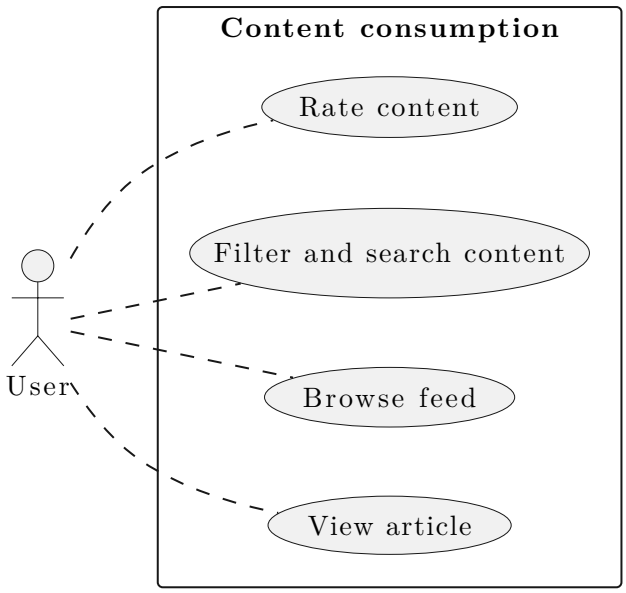

Figure 19: The use case diagram of content consumption

Usecase ID	Name	Primary Actor	Goal
UC06	View article	User	Read an article post
UC07	Browse feed	User	View personalized or global content feed
UC08	Filter and search content	User	Find content using tags, metadata, or text search
UC09	Rate content	User	Upvote or downvote post, comments, pull requests, or issues

Table 11:

3.3.2.3. Article lifecycle

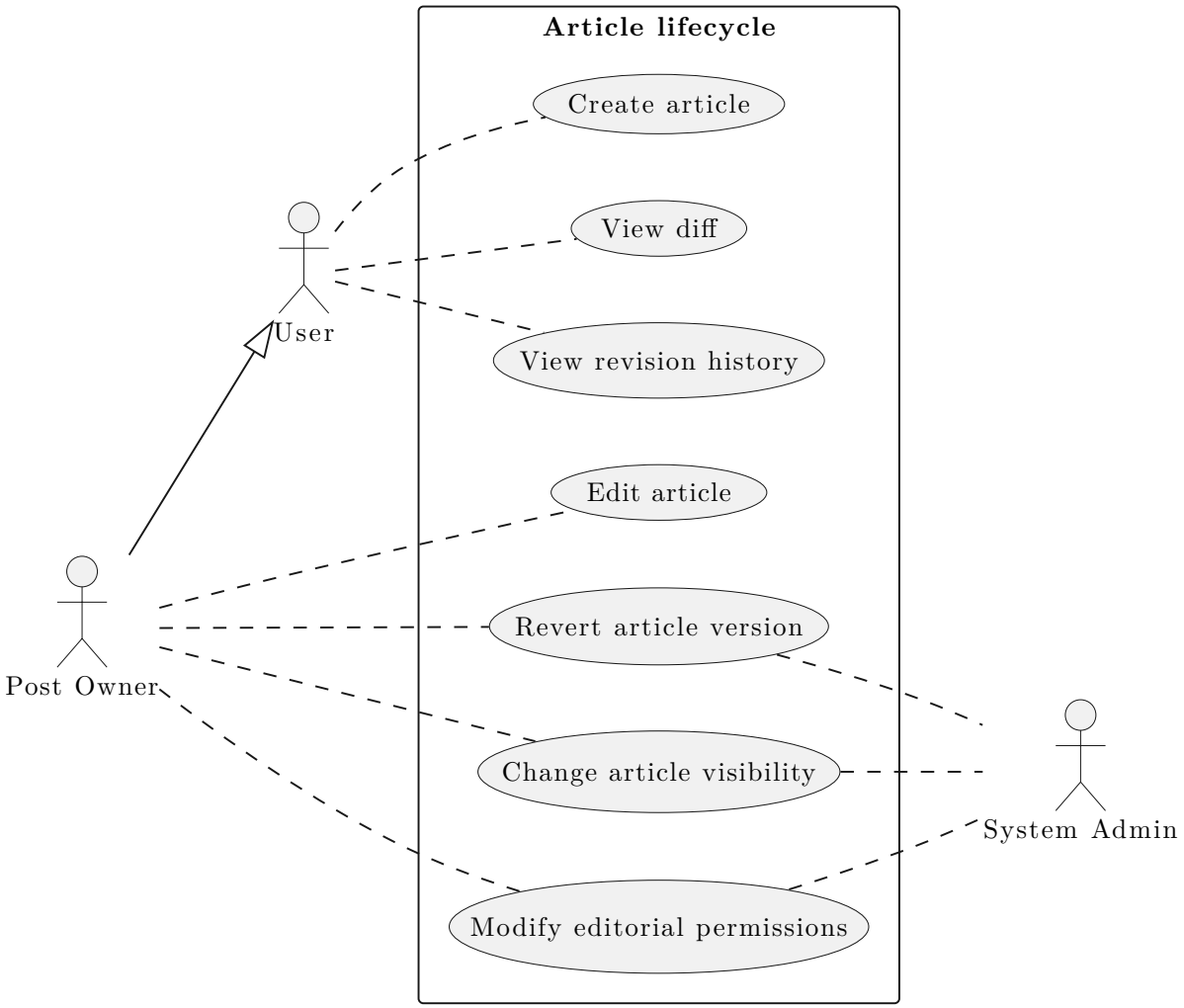

Figure 20: The use case diagram of article lifecycle

Usecase ID	Name	Primary Actor	Goal
UC10	Create article	User	Create a new article post
UC11	Edit article	User	Modify an existing article
UC12	View revision history	User	Inspect past versions of an article
UC13	View diff	User	Compare two versions of an article
UC14	Revert article version	User, Admin	Restore a previous version
UC15	Change article visibility	User, Admin	Change article visibility to common users
UC16	Modify editorial permissions	User, Admin	Modify the editing rights, or role to a specific article of another user

Table 12:

3.3.2.4. Discussions

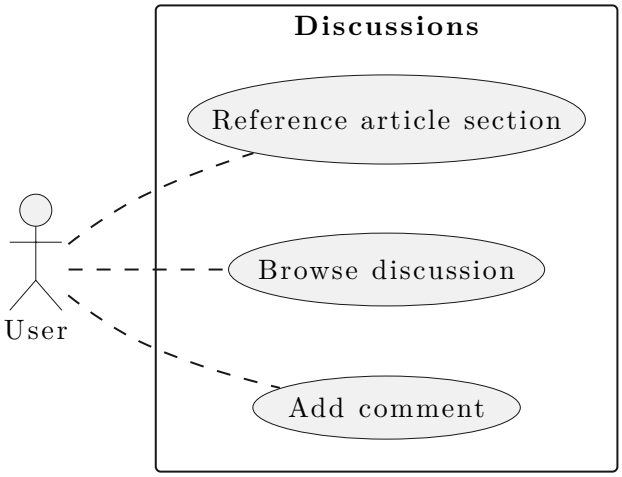

Figure 21: The use case diagram of discussions

Usecase ID	Name	Primary Actor	Goal
UC17	Add comment	User	Post a comment or reply in a discussion thread
UC18	Browse discussion	User	Navigate threaded discussions with sorting or collapsing
UC19	Reference article section	User	Reference a specific section of the article in discussion

Table 13:

3.3.2.5. Issues

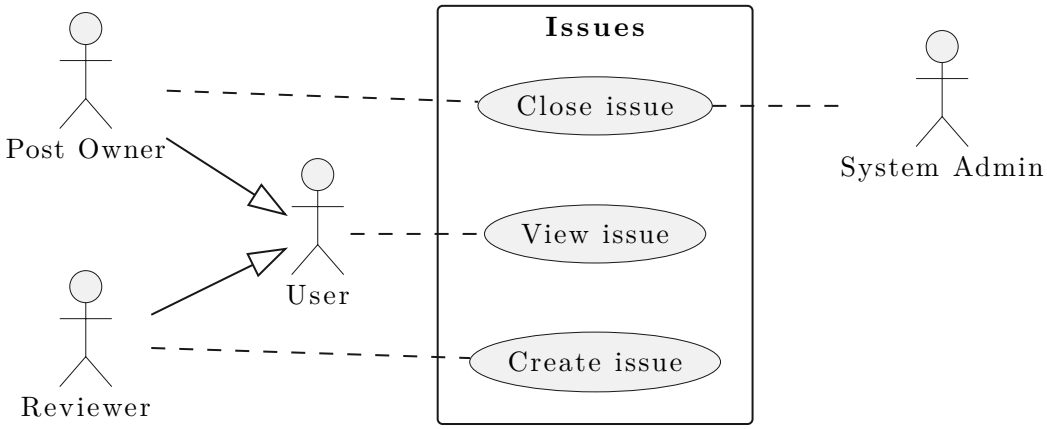

Figure 22: The use case diagram of issues

Usecase ID	Name	Primary Actor	Goal
UC20	Create issue	User	Report a problem or concern with an article
UC21	View issue	User	Read issue details and discussion
UC22	Close issue	User, Admin	Mark an issue as resolved or invalid

Table 14:

3.3.2.6. Pull-requests

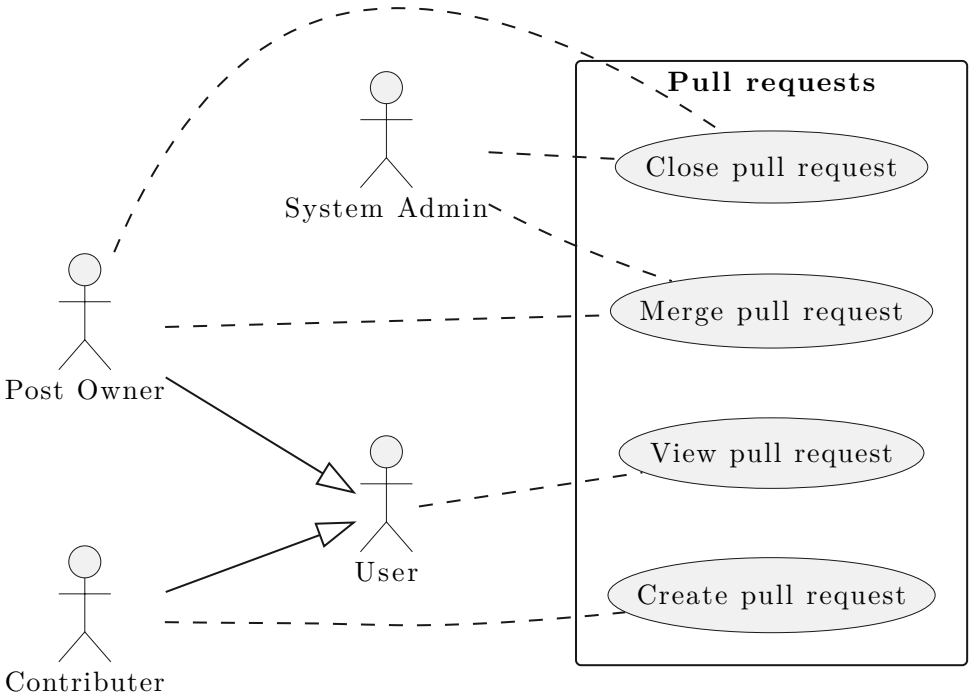

Figure 23: The use case diagram of pull requests

Usecase ID	Name	Primary Actor	Goal
UC23	Create pull request	User	Propose changes to an article
UC24	View pull request	User	Inspect proposed changes and diffs
UC25	Merge pull request	User, Admin	Accept and apply proposed changes
UC26	Close pull request	User, Admin	Mark a pull request as resolved or invalid

Table 15:

3.3.2.7. Notes

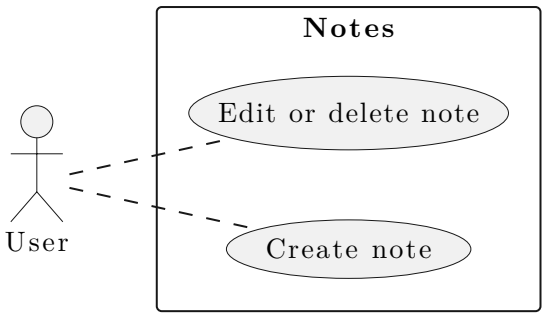

Figure 24: The use case diagram of notes

Usecase ID	Name	Primary Actor	Goal
UC27	Create note	User	Create a standalone or referenced note
UC28	Edit or delete note	User	Maintain personal notes

Table 16:

3.3.2.8. Moderation & administration

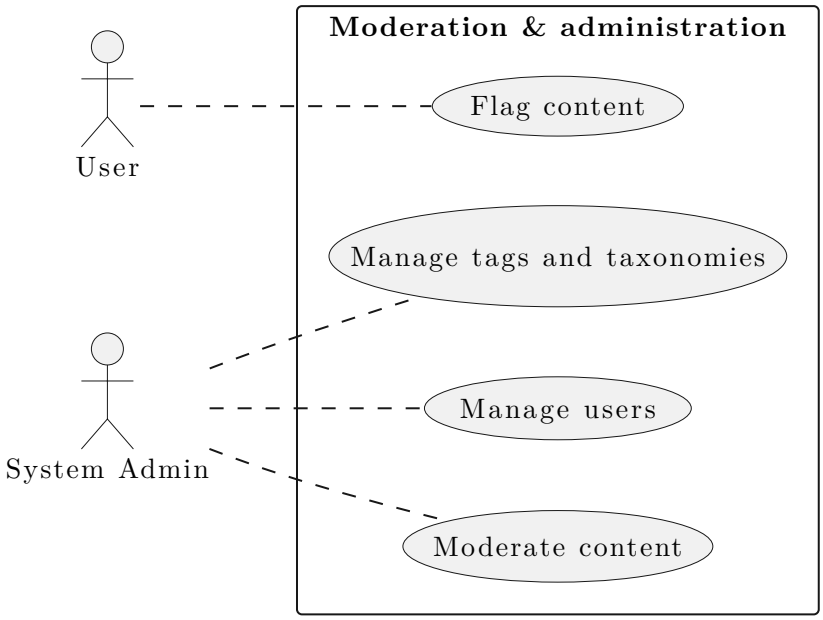

Figure 25: The use case diagram of moderation & administration

Usecase ID	Name	Primary Actor	Goal
UC29	Flag content	User	Report content for moderator review
UC30	Review content	Admin	Review, approve, or take action on flagged content
UC31	Manage users	Admin	Inspect or update user roles and permissions
UC32	Manage tags and taxonomies	Admin	Maintain global classification structures

Table 17:

3.3.3. Usecase description

UC01	**Pull Request**	
Dependencies	Article	
Description	This use case describes how a user proposes, views, edits, and finalizes changes to an existing article through a pull request workflow. A pull request encapsulates proposed modifications, supports review via diffs and discussions, and may be approved, edited, or abandoned by its author.	
Precondition	The user is authenticated and has permission to propose changes to the target article. The target article exists and is accessible to the user.	
Ordinary Sequence	**Step**	**Activity**
	1	The user navigates to the pull requests list.
	2	The user optionally applies filters, sorting, or pagination to browse pull requests.
	3	The user selects “create pull request” and enters the pull request editor.

UC01	**Pull Request**	
	4	The user edits proposed changes and may preview the rendered result.
	5	The user confirms creation, and the pull request is created and stored.
	6	The user may later view an existing pull request.
	7	The user may inspect the pull request post, associated comments, or the list of changes (diffs).
	8	The author may edit the pull request and confirm the updated version.
	9	The author may approve the pull request, concluding the process.
Post Condition	A pull request is created, updated, or approved. Its state and associated discussions and diffs are persisted and become visible according to permissions.	
Exceptions	**Step**	**Activity**
	1	The user cancels pull request creation, and no changes are saved.
	2	The user lacks sufficient permissions, and the action is denied.
Comments	This use case focuses on the author-side lifecycle. Administrative approval, rejection, or merging is covered in a separate moderation or merge use case.	

Figure 26: Pull request usecase description

3.4. System architecture

The architecture of BKHack was developed with the strategic goal of minimizing communication layering while balancing expressiveness and performance. Determining the system structure requires analyzing the key requirements to identify the architectural characteristics that are most critical for the system's success, and translating those constraints into an appropriate architectural style.

3.4.1. Analysis of architectural styles

Before defining the architectural style, it is essential to identify the characteristics that drive the system's design. For BKHack, a university-integrated social news platform centered on community-sourced knowledge and rigorous revision control, three characteristics emerge as essential for long-term effectiveness and sustainability: modularity, cost effectiveness, and integrity.

Modularity refers to the degree to which the software is composed of discrete, independent units. For BKHack, high modularity is crucial because the system integrates distinct functional areas, including user accounts, content posting, complex pull request/revision management, and voting/ranking,. Organizing the codebase into independent components helps support maintainability and extensibility, allowing future integrations or new features to be added without necessitating a full system redesign. Furthermore, achieving high modularity is key to meeting the non-functional requirement that the codebase be organized into modular components and that each major subsystem be implemented as an independent module,.

Cost effectiveness dictates that the system must remain economical in terms of development complexity, infrastructure requirements, and maintenance overhead. Given that the platform is designed for internal use within a university, and as a capstone project instead of a commercial product, it does not require the massive, high-latency scalability often associated with globally distributed applications,. Architectural decisions that prioritize simplicity and operational directness naturally keep the overall costs low, which is a necessary trade-off to ensure the system's viability,.

Integrity as a sub-characteristic of security, is defined by the degree to which the software prevents unauthorized access to or modification of software or data. For BKHack, integrity is paramount for maintaining academic credibility. The core feature of the platform is its version-controlled, fact-checking revision and pull request system, which demands that all changes (merges, edits, ownership transfers) be logged for accountability (FR20) and that every edit creates a new, persistently stored version (FR4). Therefore, the architecture must guarantee data validity and consistency in all operations, especially concerning sensitive data like user credentials, which must be hashed and protected (NFR9).

To determine the most suitable architectural style for BKHack, we evaluated candidate architectures against the prioritized characteristics: Modularity, Cost Effectiveness, and Integrity. We considered the core trade-off between monolithic and distributed architectures, recognizing that while monolithic approaches offer simplicity, distributed architectures provide significant gains in performance and scalability.

The layered architecture is a monolithic style defined by organizing components into logical horizontal layers e.g. a set of presentation layer, business layer, persistence layer, and database layer. This style is fundamentally technically-partitioned, grouping components by technical role rather than domain.

Pros:

- Cost effectiveness: This style ranks highly in simplicity and low in cost, making initial development simple and suitable for prototypes.
- Testing is easier because all components run together in a single environment.
- Deployment overhead is low, as the entire system is deployed as one unit.

Cons:

- Modularity: This style scores low in modularity, maintainability, and evolvability.
- Domains are spread across technical layers, which challenges domain-driven design.
- Integrity: The system has a single point of failure, meaning a crash in one part can bring down the entire system, negatively impacting reliability.
- Maintenance is difficult since small changes require rebuilding and redeploying the whole system.

Microservices is a distributed style where each service runs in its own process, embodying high decoupling by physically modeling the logical notion of bounded context. Services are typically fine-grained.

Pros:

- Modularity: The architecture prioritizes and achieves extremely high scores across maintainability, testability, and evolvability due to its decoupled nature.
- Faults are isolated, meaning failures are contained within a single service, contributing to high fault tolerance.
- It supports parallel development, as different teams can work on separate services simultaneously.

Cons:

- Cost Effectiveness: This style ranks lowest in simplicity and highest in cost due to the complexity of system design, deployment, and the high infrastructure required to maintain and monitor multiple services.
- Integrity: Extreme decoupling in this architecture style often leads to challenges with transactional coordination across services. Unlike monolithic systems that rely on ACID guarantees, microservices typically rely on BASE transactions (eventual consistency), which sacrifices data integrity for performance and scalability.
- Network calls are required for communication, which introduces latency and increases operational complexity.

The service-oriented architecture (SOA) is a practical, distributed style recognized for its architectural flexibility, which is achieved through a distributed macro layered structure. It consists of separately deployed coarse-grained domain services and typically utilizes a centrally shared monolithic database.

Pros:

- Integrity: SOA maintains ACID transactions better than other highly distributed architectures because its coarse-grained services use regular ACID database

transactions involving commits and rollbacks to ensure database integrity within a single domain service.
- Modularity: Services are domain-partitioned, resulting in good scores for maintainability, testability, and evolvability. This structure offers agility and allows future integrations without full redesign.
- Cost Effectiveness: This style avoids the high complexity and cost associated with fine-grained distributed architectures like microservices. It ranks highly in simplicity and low in cost relative to other distributed approaches.
- It offers high fault tolerance because services are separated, isolating faults to a single unit.

Cons:
- The centralized database, although beneficial for integrity, may pose partitioning challenges in the future.
- It scores lower in scalability and elasticity compared to microservices due to the coarse-grained nature of its services.
- The architecture introduces complexity in deployment compared to monolithic styles, requiring load-balancing capabilities if multiple service instances are needed.

The layered architecture fails to meet the crucial high Modularity and Maintainability goals required.

The microservices architecture achieves maximum modularity but is disqualified due to its excessive Cost and complexity, which conflicts with the goal of Cost Effectiveness for a university-integrated project. Furthermore, microservices' reliance on eventual consistency complicates the assurance of Integrity for critical versioning and transactional data.

The service-oriented architecture style provides the optimal balance required for BKHack: it achieves good modularity and excellent fault tolerance, while maintaining the crucial ACID transactions required for high integrity of the revision control system. Crucially, it accomplishes these benefits at a significantly lower cost and complexity relative to microservices, making it a viable and pragmatic choice for the current scope.

3.4.2. Conclusion

The service-oriented architecture architectural style is a logical choice for architecting the BKHack system. During the development of BKHack, we also incorporate elements from other architectural styles, resulting in a hybrid architecture. Currently, BKHack is a multi-tier service-oriented architecture,

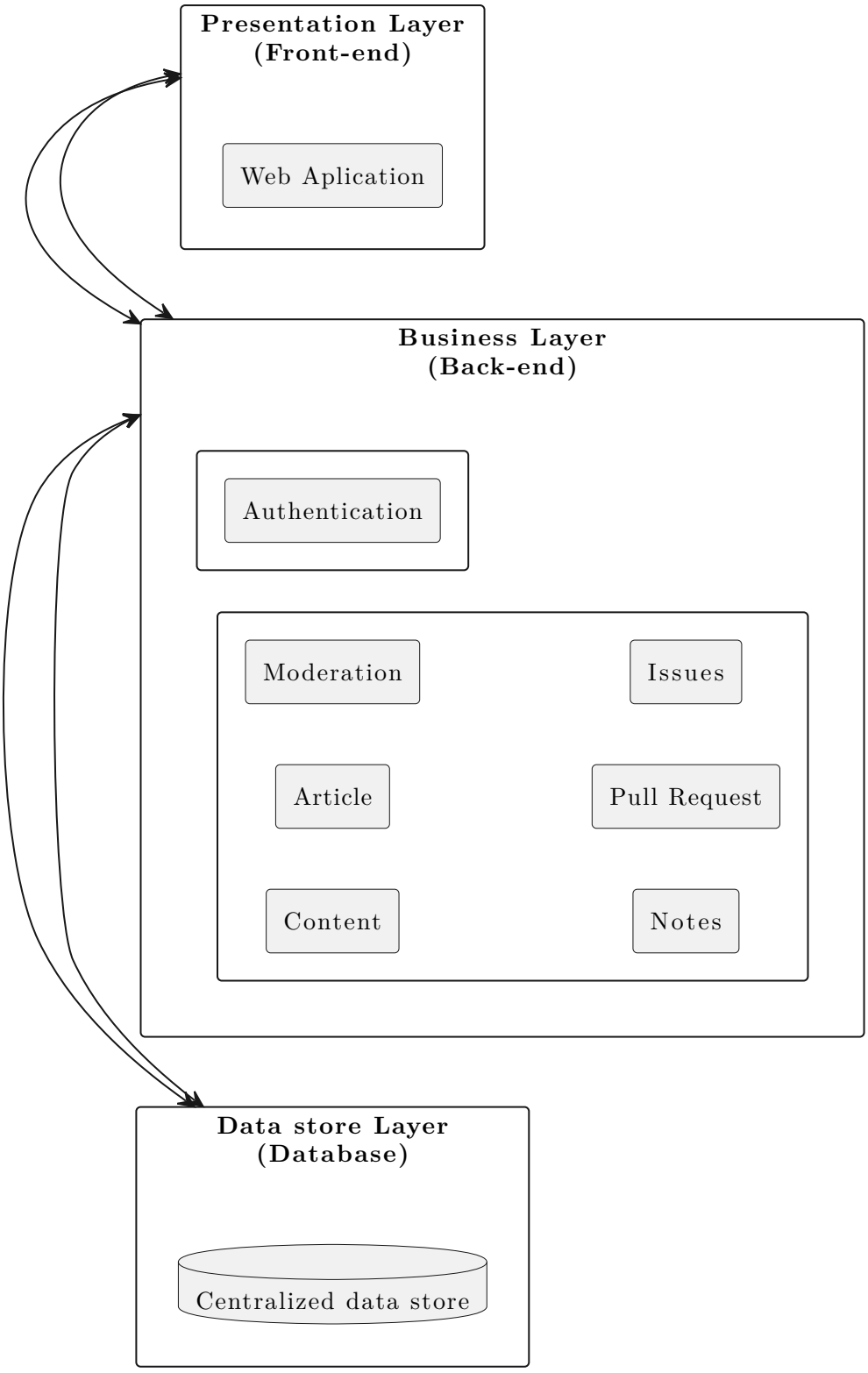

Figure 27: System architecture of BKHack

3.5. Entities and relationships

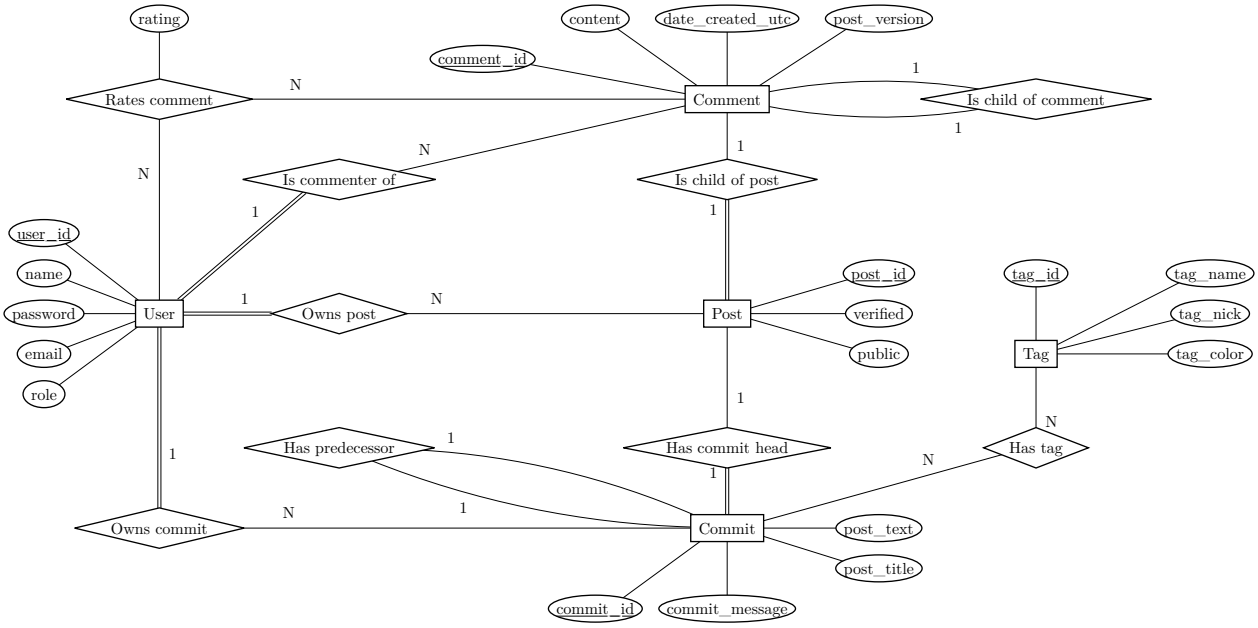

Figure 28: Concept ER Diagram, in Chen's notation, for the entities and relationships in the BKHack system

3.6. Architecture of the front-end

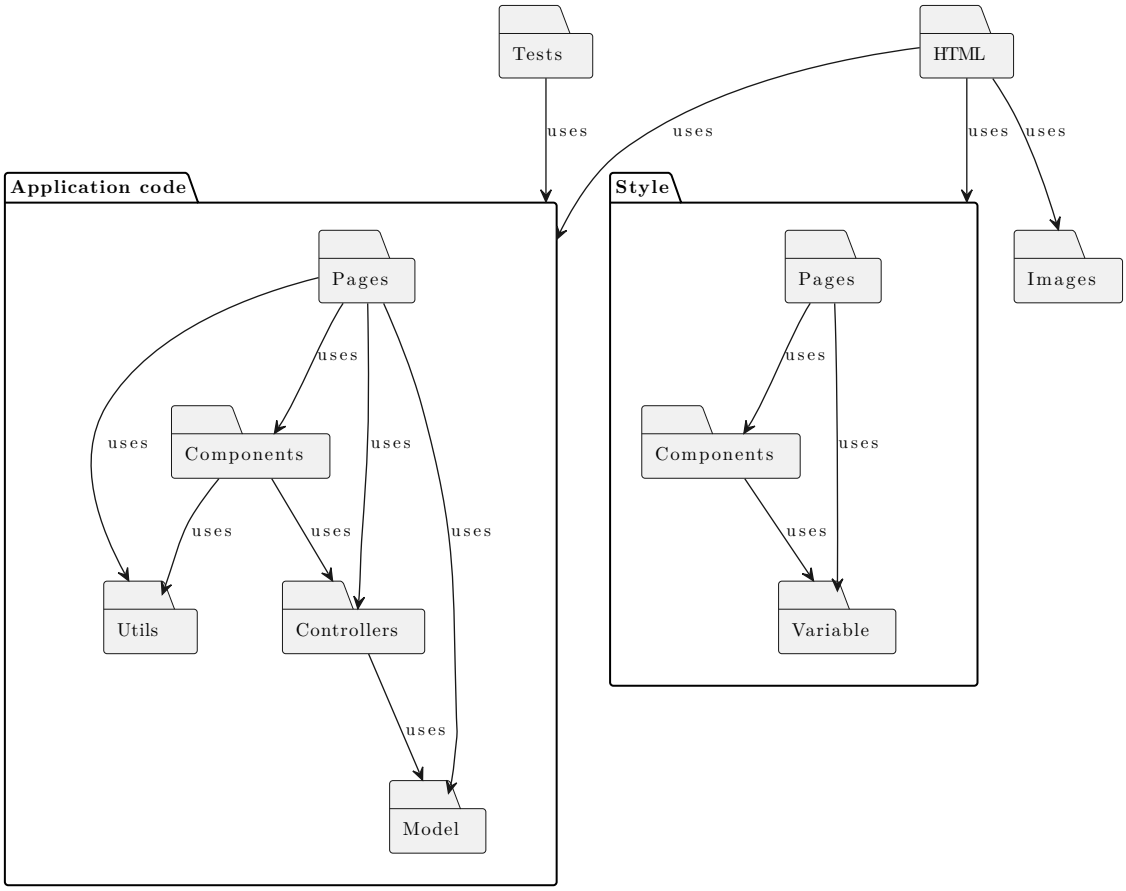

Figure 29: Module view of the front-end system

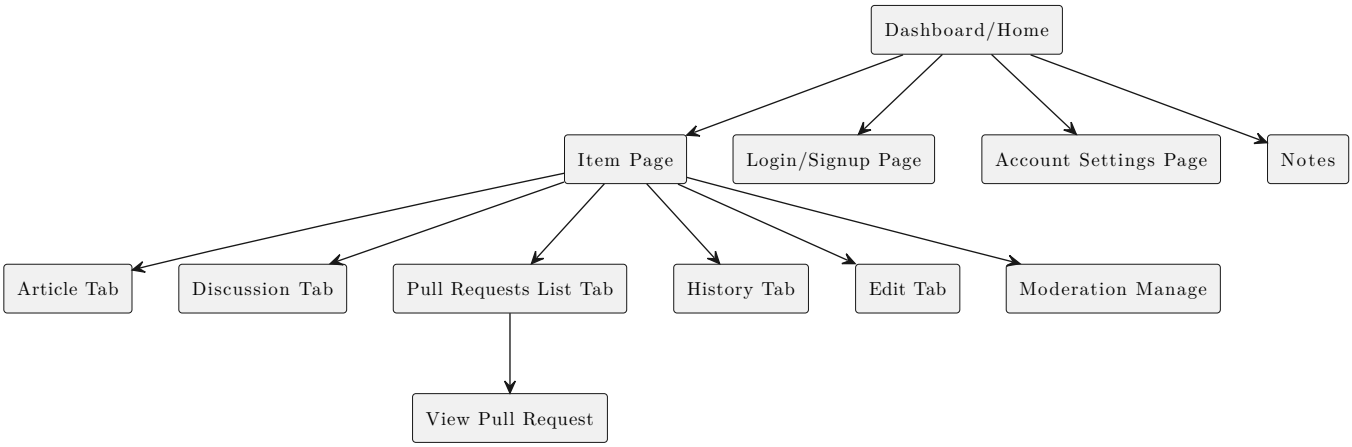

Figure 30: Sitemap view of the front-end system

3.7. States and interactions of the front-end

This section details the state management and user flow logic for the BKHack front-end.

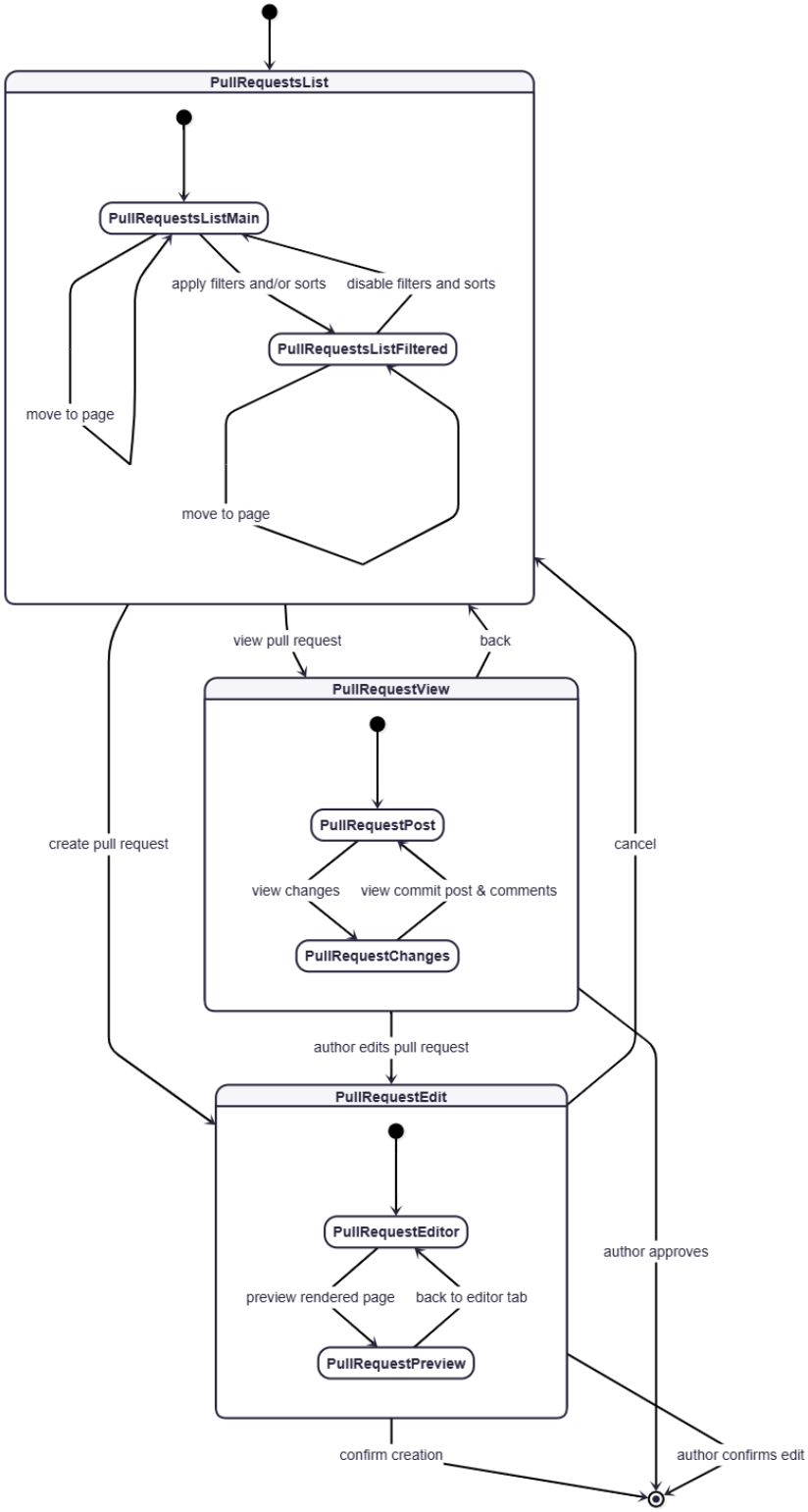

Figure 31: State diagram for user flow during a pull request session

This state diagram describes the user flow and state transitions involved in a pull request session on the BKHack front-end. The interaction begins at the Pull Requests List, which serves as the primary entry point for browsing existing pull requests. Within this list, users can paginate through results and optionally apply or remove filters and sorting criteria, with pagination supported in both the filtered and unfiltered states.

From the list view, users may initiate the creation of a new pull request, transitioning into the Pull Request Edit state. This editing state internally manages two substates: an editor for composing or modifying content, and a preview for viewing the rendered result. Users can freely switch between editing and previewing before confirming creation or edits. Confirmation exits the flow, while cancellation returns the user to the pull request list.

Users may also open an existing pull request from the list, entering the Pull Request View state. This view allows navigation between the main pull request post and its associated changes, such as commits and diffs, while preserving the ability to return to the list. From this state, the author may approve the pull request, terminating the session, or transition back into the edit flow to modify the pull request before confirming the changes.

3.8. User interface wireframing

3.8.1. Analysis

To determine the layout of BKHack, we first analyzed the specific behavioral patterns of our target demographic: HCMUT computer science students and lecturers. Survey results indicated that this audience values citation and credibility factors, often struggling with the fragmented and outdated academic discourse currently dispersed across various informal platforms.

This technical audience is accustomed to the high information density and functional efficiency found in IDEs, technical documentation, and command-line interfaces. Consequently, we determined that BKHack should not be modeled as a marketing-oriented site or a standard social media platform, but rather as an academic knowledge management and dissemination system for those who build and refine ideas. This realization led us to reject “marketing-heavy” or “mobile-first” layouts in favor of a desktop-first, left-aligned structural approach.

Human reading patterns in technical documentation typically follow a left-to-right scanning habit; therefore, we established that all critical content must be strictly left-aligned to optimize for F-pattern scanning behavior. With these constraints, we arrived at a two-column layout system featuring a 70/30 split.

The primary 70% area is dedicated to the main content to ensure the “current truth” of a topic remains the focal point, while the remaining 30% is occupied by a toggleable right-side sidebar for supplementary context, allowing users to process primary information before seeking technical metadata. The sidebar is toggleable for the option to minimize distractions and make use of screen real estate when it is needed.

3.8.2. Demonstration

The following prototype visualizations demonstrate this layout by stripping away all icons and text elements to showcase the underlying structural hierarchy.

3.8.2.1. Post feed layout prototype

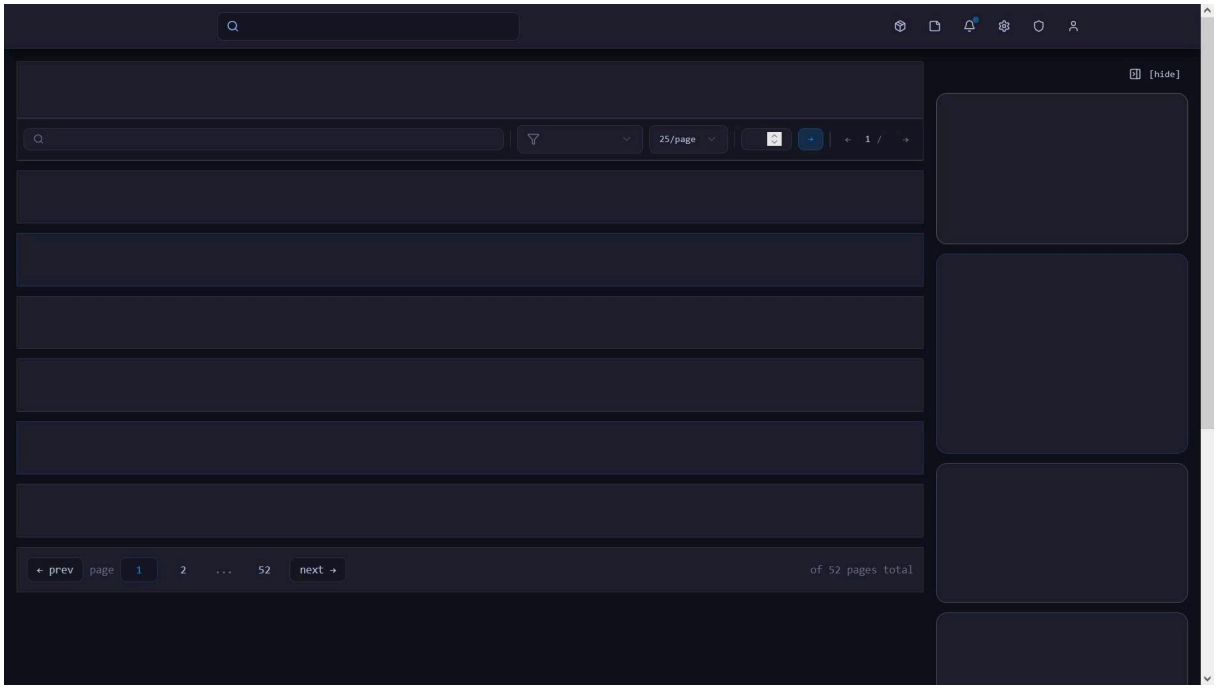

Figure 32: The post feed layout of BKHack

The feed is structured to maximize vertical information density, favoring a structure-first layout that mirrors a repository or a wiki rather than an endless scroll.

- Top Navigation Container: A persistent, sticky horizontal bar at the top provides global context and search access regardless of scroll depth.
- Filter and Sort Row: Situated immediately above the content, this block serves as the primary tool for narrowing the scope of information without leaving the page.
- Minimal-Gap Content List: Inspired by GitHub, the primary area consists of stacked rectangular blocks with tight vertical gaps. This reduces visual noise and allows the eye to travel through a large volume of items without interruption.
- Toggleable Right Sidebar: A dedicated block for non-essential navigation that can be collapsed to give content the full width on smaller screens.

3.8.2.2. Post view layout prototype

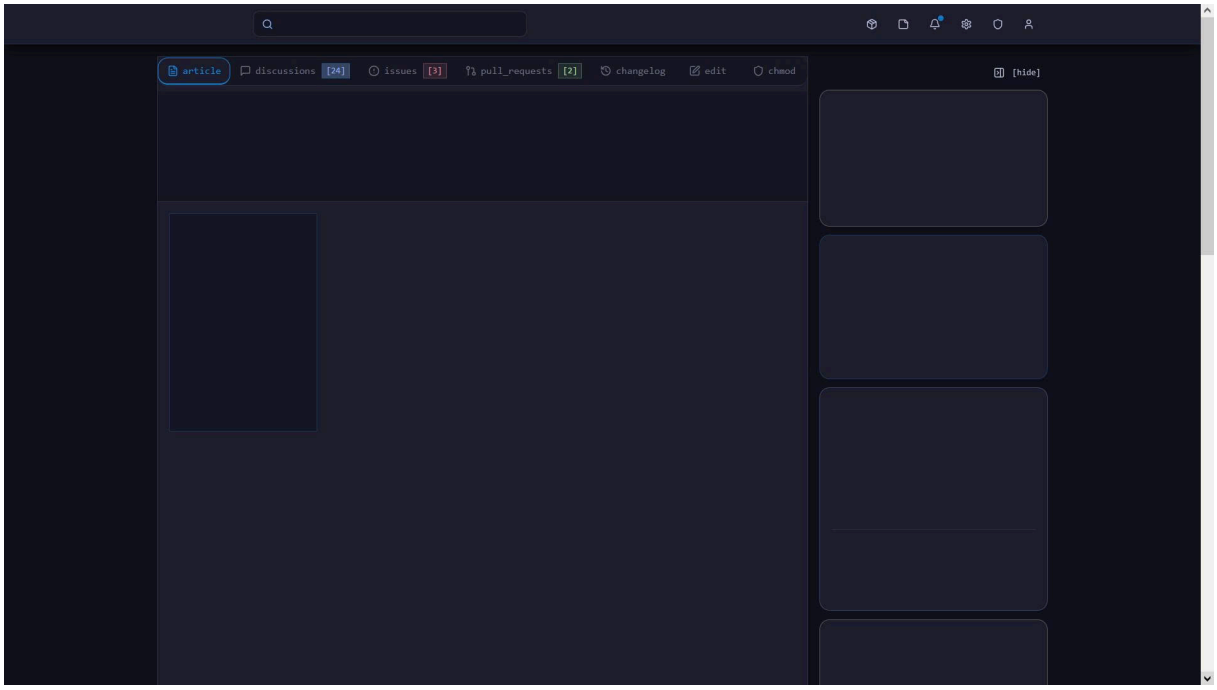

Figure 33: The post view layout of BKHack

The view page transitions to a document-centric hierarchy, mimicking a structured documentation of an open-sourced software product.

- Header Section: A prominent top block dedicated to essential identifiers to establish the subject immediately. This item is common among the containers for quick identification of the current page.
- Tabbed Navigation Bar: Directly beneath the header, a horizontal row of containers allows users to navigate the multiple aspects of a post, such as articles, discussions, and history without overwhelming the user on a single page.
- Main Body Area: A centered container with a reduced max-width compared to the feed, ensuring a comfortable line length for long form reading.
- Sidebar Metadata Blocks: Smaller containers on the right provide summary status updates and quick actions, keeping them accessible but separate from the core text.

3.9. User interface prototyping

3.9.1. First minimum viable product

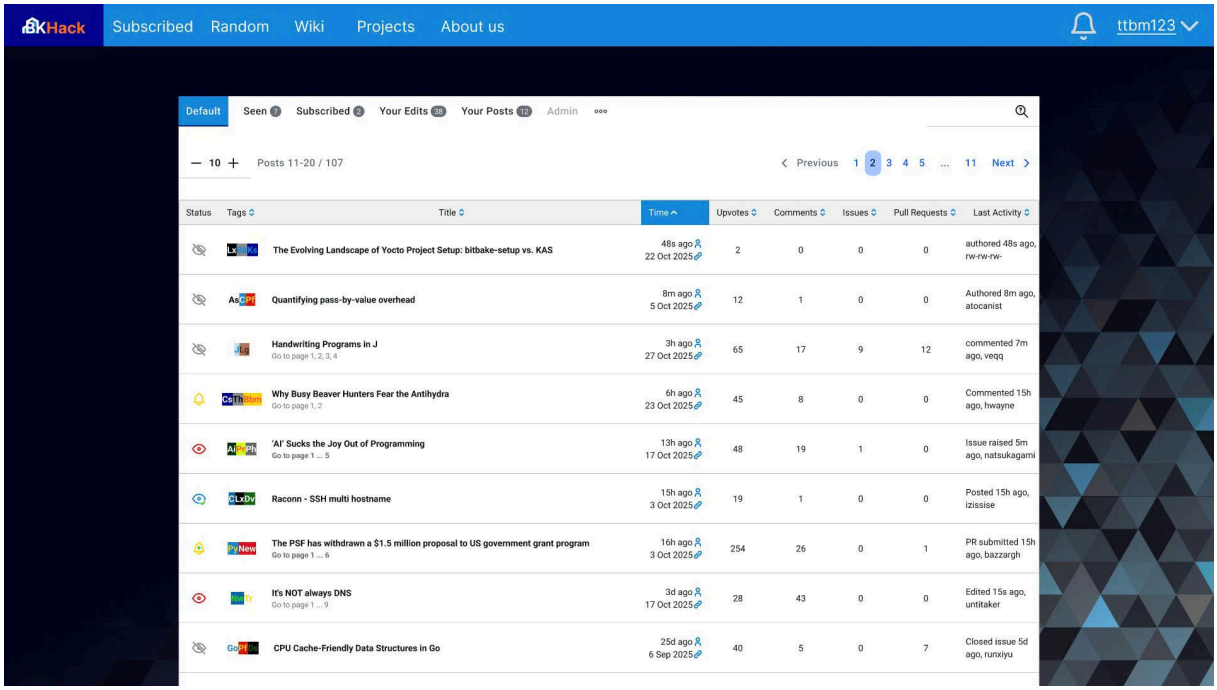

Figure 34: The initial prototype of BKHack

The initial stage of BKHack's development involved a wireframe prototype built using Figma. While this environment served its purpose for design validation, the we admits that the earliest iterations were significantly flawed in their execution of the project's core identity consisting of but not limited to:

- Sterile Aesthetic: The first version was criticized too bland in its use of colors and having neither lack of depth nor sense of hierarchy.
- Lack of Originality: Early designs looked too similar to existing platforms like its direct inspiration HackerNews and Lobste.rs. While these are clean, they did not outwardly reflect the "university CS department intranet" theme the team desired.
- UI Annoyances: Experimental features, such as extreme symbolic notation (e.g., `24c 3i 2pr` for comments and issues), were found to be "annoying to use" and potentially confusing even for initiated users. Overall, it did not deliver on our intended design.

Influence on Later Decisions: These early "failures" were instrumental in shaping the current Design System. The realization that the prototype was too derivative led the team to adopt a Terminal-UI (TUI) aesthetic—specifically inspired by modern tools like the Ghostty terminal—an existing aesthetic that not only fit are criterias, but also ties in quite well thematically.

3.9.2. React prototype

Following the initial design phase in Figma, the we developed a second prototype using React, TypeScript, and Tailwind CSS. While the Figma iteration allowed for

rapid visual experimentation, the tools provided were rather restrictive in its usage. This second prototype was created to apply the lessons learned previously and decide on a more concrete design.

Despite admitted still being a rough approximation of the final product, it successfully validated several core design pillars, and further presented the design under more rigorous scrutiny.

Ultimately, while the final production application will be implemented in another framework, but this React prototype remains the primary reference for the visual and structural rules that define the BKHack experience.

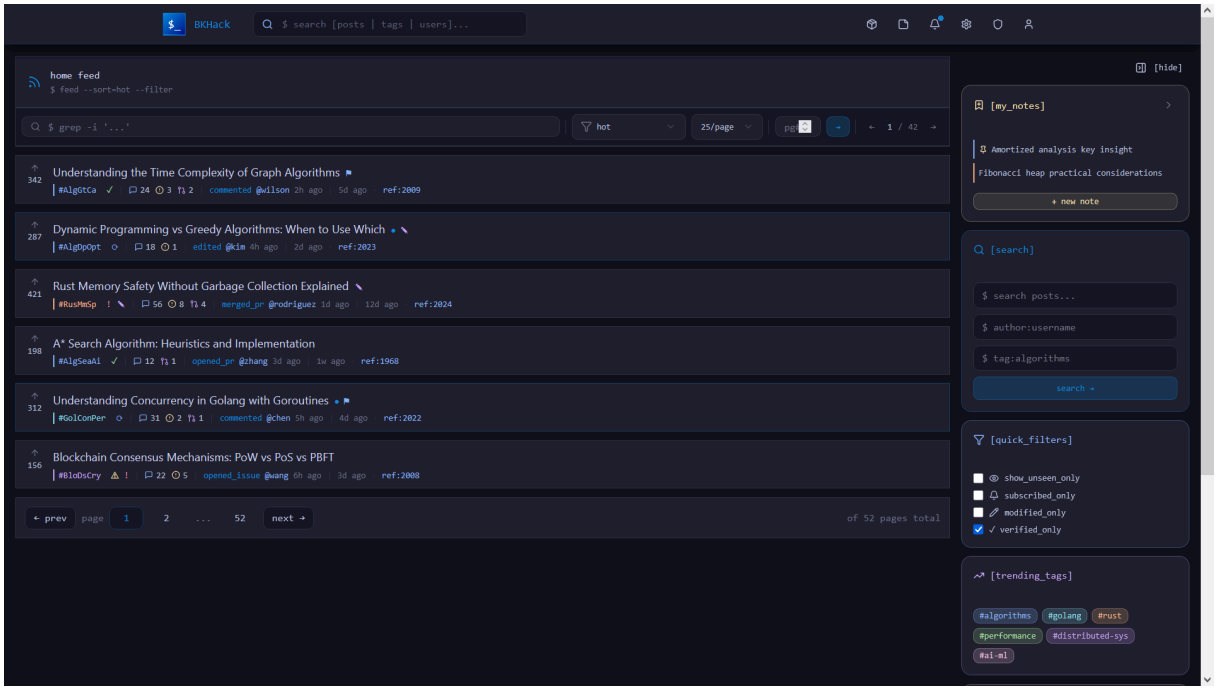

Figure 35: The post feed prototype of BKHack

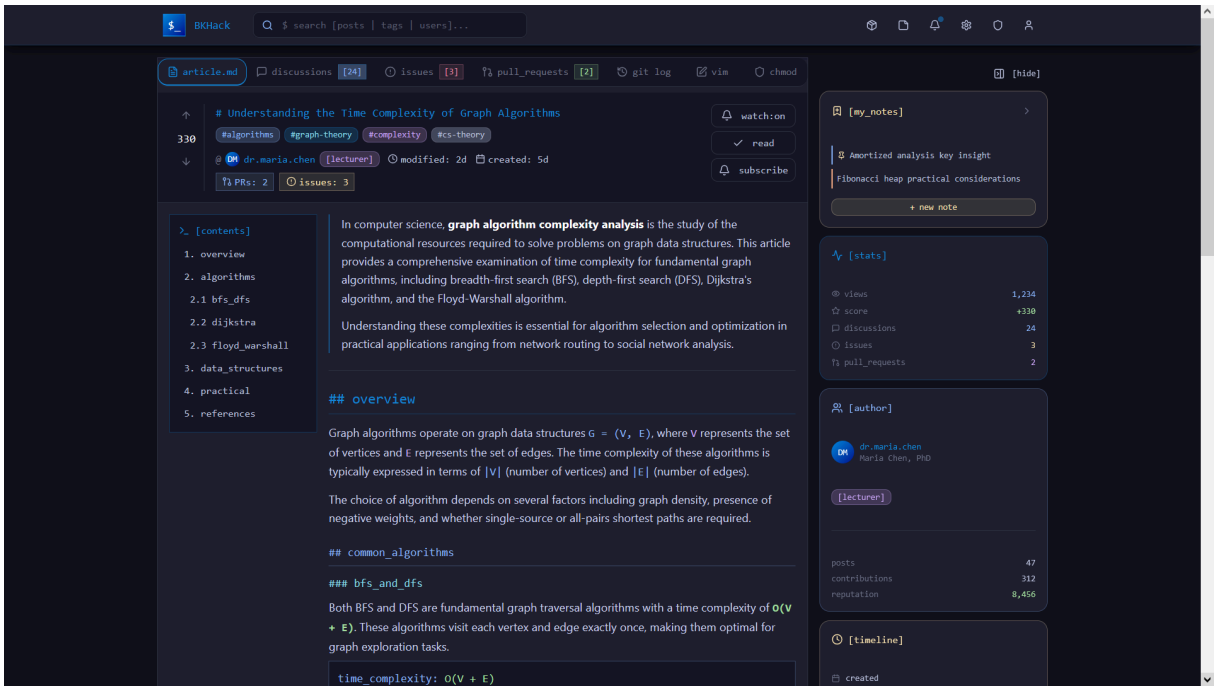

Figure 36: The article view prototype of a post in BKHack

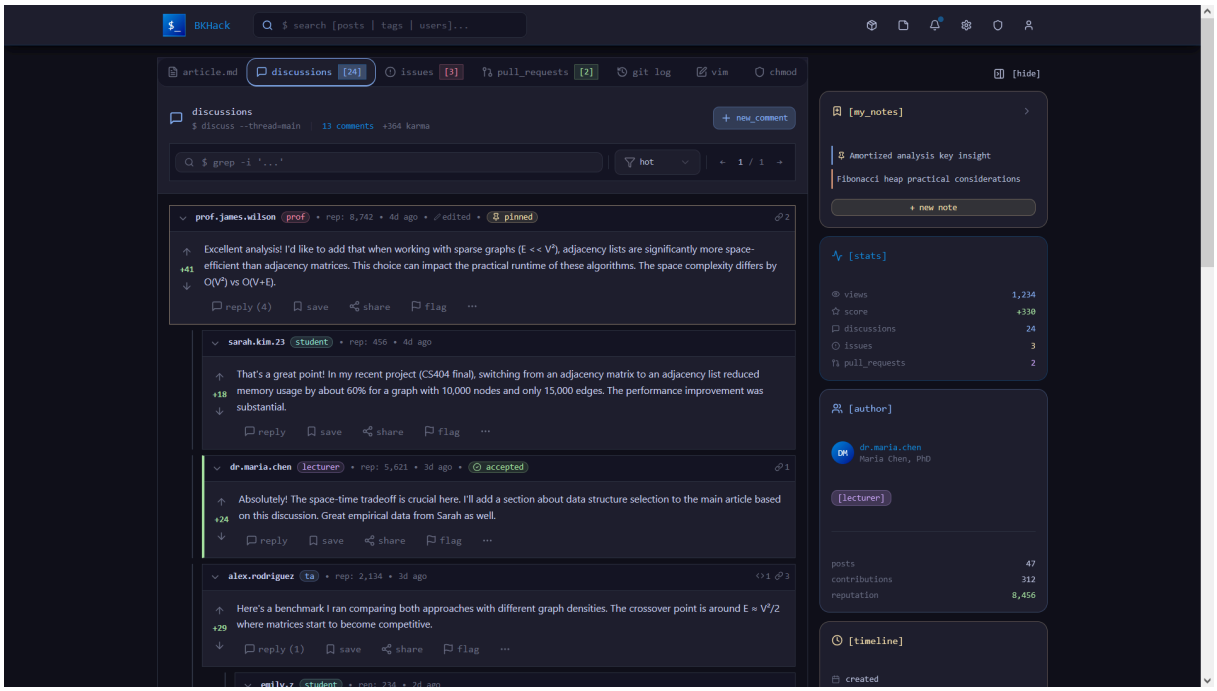

Figure 37: The discussion view prototype of a post in BKHack

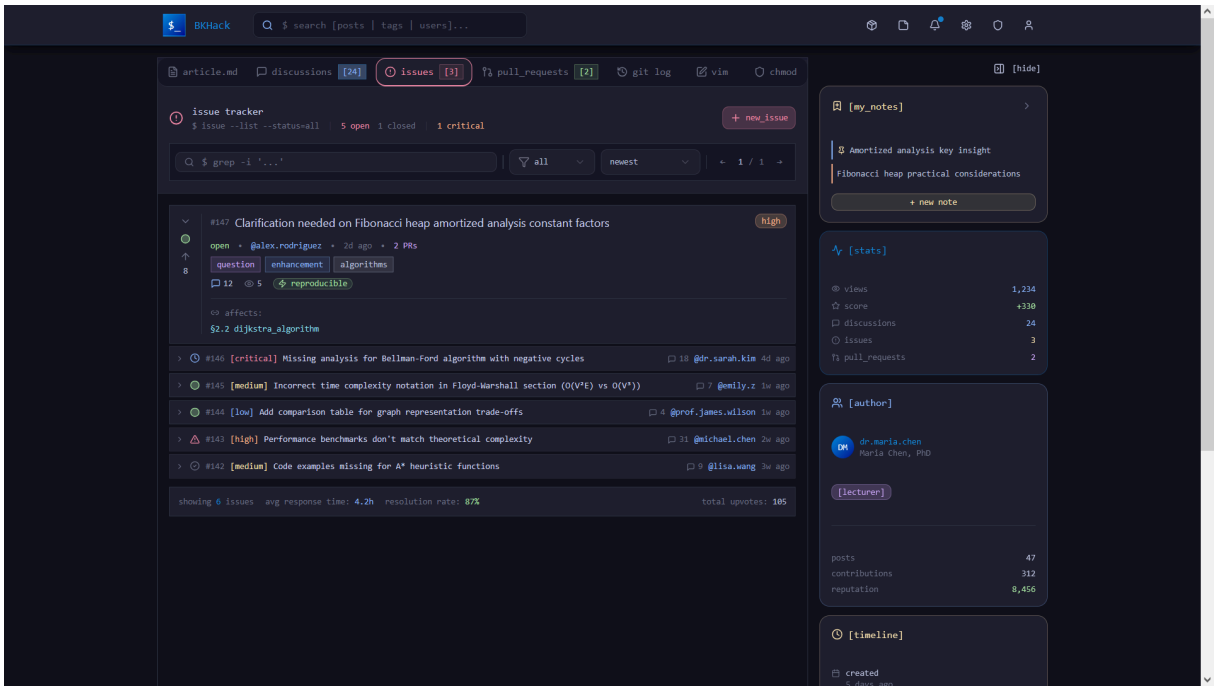

Figure 38: The issue view prototype of a post in BKHack

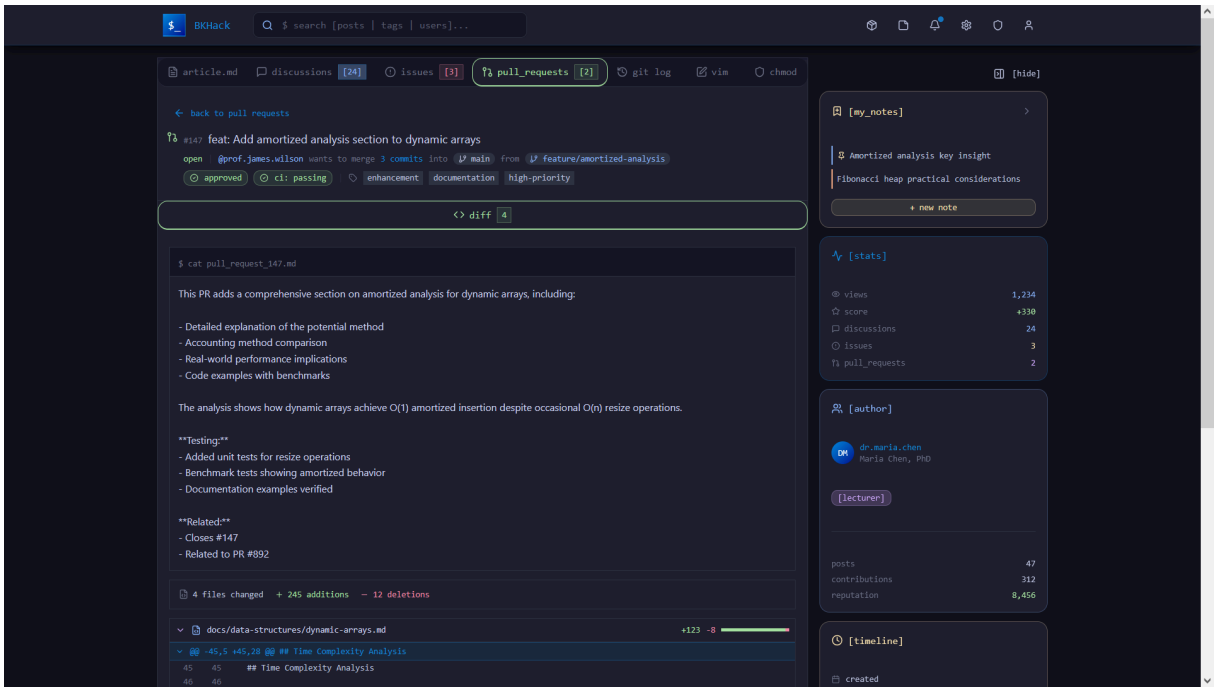

Figure 39: The pull request view prototype of a post in BKHack

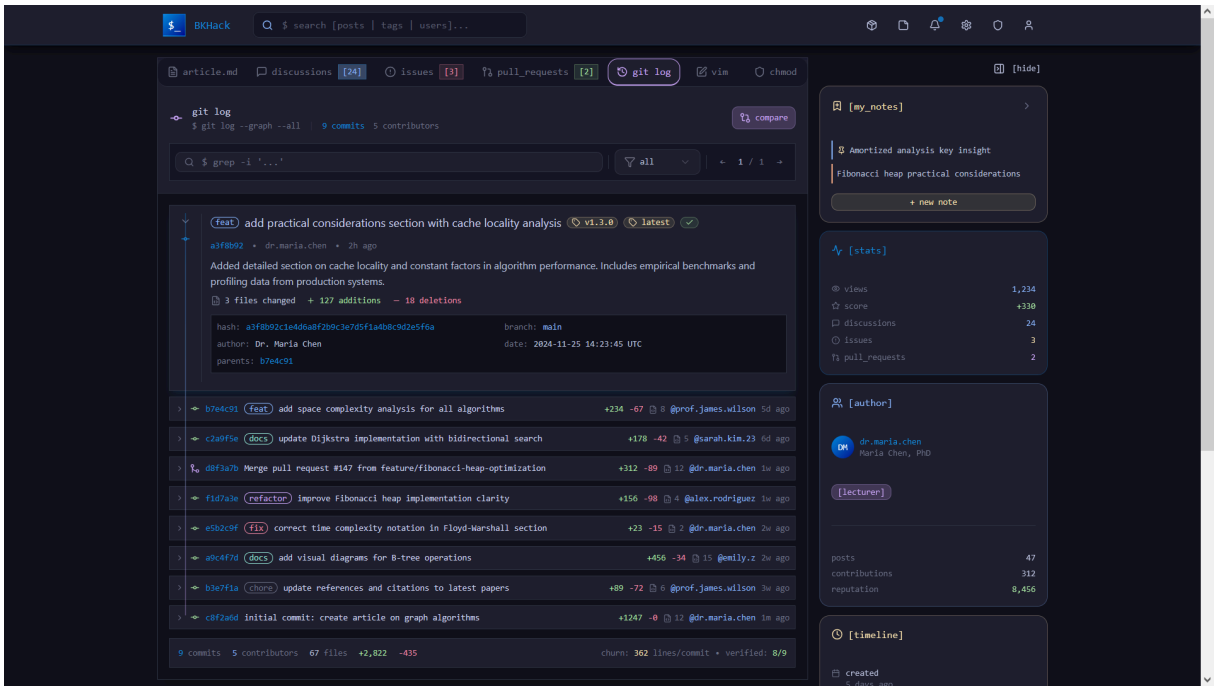

Figure 40: The history view prototype of a post in BKHack

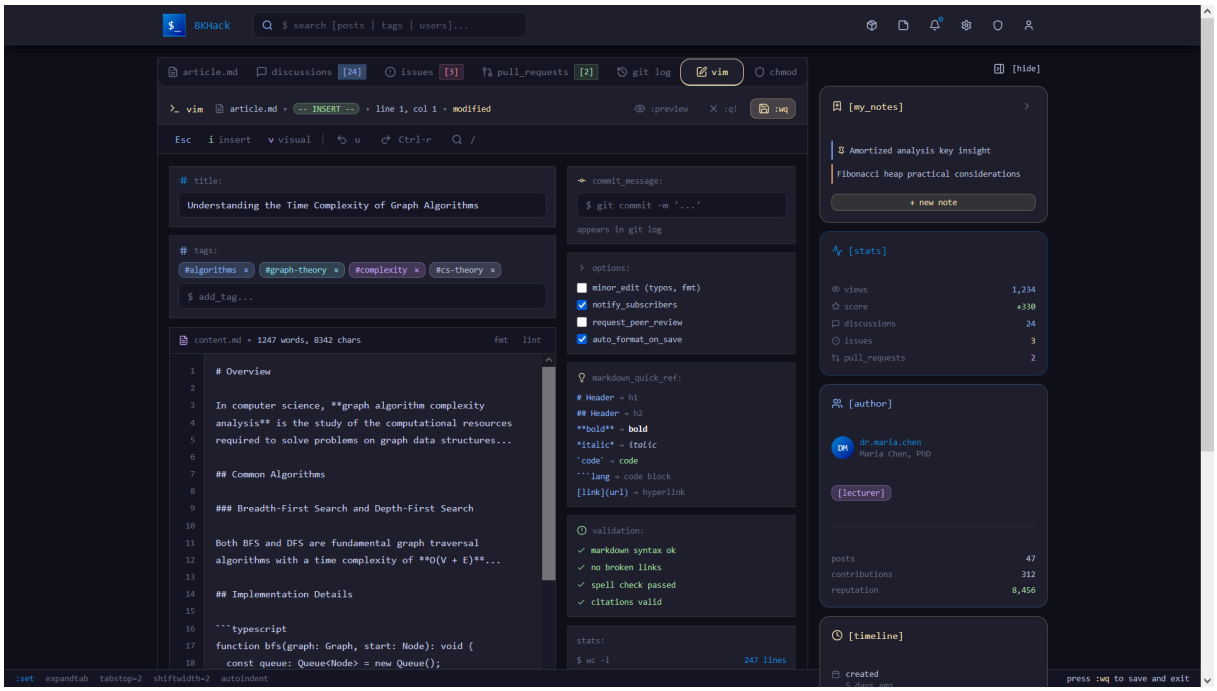

Figure 41: The edit view prototype of a post in BKHack

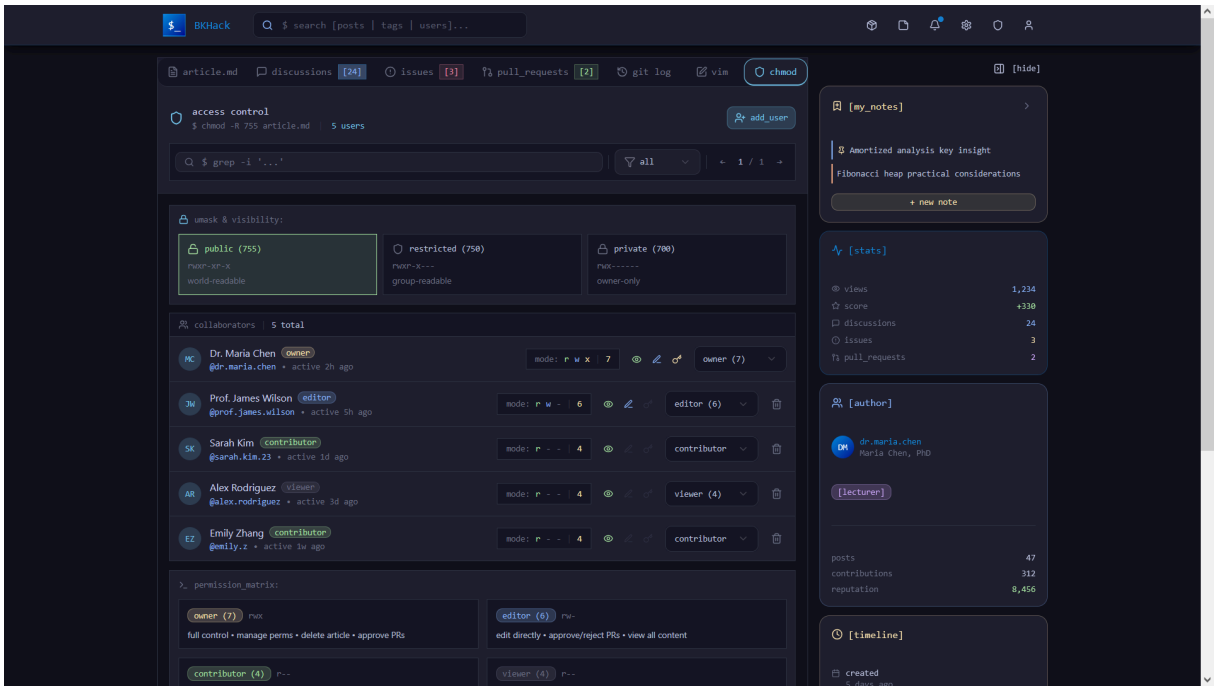

Figure 42: The permissions view prototype of a post in BKHack

4. Realization

4.1. Technologies

4.1.1. Front-end

For languages and frameworks, there are metric - certain attractive features that make the language more robust and pleasant to develop web UI applications. We have developed a collection of metrics which are:

- Whether or not the technology supports inline writing of HTML or HTML-like markup or other forms of markup.
- How strict is the type system and its programming experience. Static type-checking is preferable.
- How well supported documentation facilities. This is mentioned as a design goal in Section 3.1, we want great maintainability.
- Does it come with an opinionated programming paradigm? Today there are many programming paradigms, such as object-oriented programming, functional programming, procedural programming, etc.
- Continuation of the above point: does it come with an opinionated UI programming paradigm? Today there are manyy UI programming paradigms, such as reactive programming [11], the Elm Architecture [12], immediate-mode graphics rendering [13]. These programming models specialize in the domain of UI programming and will help us build safer systems by naturally leaning developers towards intuitive mental models during programming.

Figure 43: In clockwise order from top-left: logo for the OCaml programming language; logo for the Reason programming language; logo for the Reason React binding; logo for the React library

Reason [14] (formerly ReasonML) is a functional programming language. It belongs in the language family of ML, specifically as a direct descendent of OCaml [15], a formally-proven industrial-strength functional programming language with a robust module system.

Most importantly, Reason has first-class interop binding with the React library [16], a popular system for writing UI in the reactive programming pattern. The interop, called Reason React [17], leverages Reason's strong functional features such as algebraic data types, exhaustive pattern matching, a module system as inherited from OCaml for code organization and transformation, combined with React's mature reactivity system, and a natural foreign-function interface (FFI) layer so that Reason can interact with JavaScript and thus React.

Figure 44: From left to right: logo for the Haskell programming language; logo for the Elm programming language

Elm [18] is a pure functional programming language. It belongs in the language family of Haskell [19]. Elm is known for pushing for a unique opinionated UI programming model called the The Elm Architecture, or TEA [20]. Elm also benefits from its Haskell heritage where there is a separation between pure function evaluation and monadic evaluations that can have side effects, thus a formalized way for programmers to track what their programs can and cannot do.

	JavaScript with ReactJS	**OCaml with Bonsai**	**Reason**	**Elm**
Prog. paradigms	Object-oriented Functional-ish Imperative	Object-oriented-ish Functional	Object-oriented-ish Functional	Functional
UI paradigms	FRP	FRP	FRP	TEA
Inline markup	Native (JSX)	Non-native (PPX)	Native (JSX-like)	None
Documentating	JSDoc	`.mli` files	`.rei` files	Elm Doc. Format
Type-checking	Dynamic	Static	Static	Static

Table 18:

In general, most languages and frameworks in this comparison provide some sorts of inline mark-up writing.

Similarly, all languages encourage documentation through their own textual formats. In particular, OCaml and Reason sport their own system of self-documenting signature files, where a supposedly header file for machine to parse as API also serves as documentation for humans to read

Regarding type-checking, JavaScript is the only language which doesn't fit all criteria due to being dynamically typed by default.

Combined these with personal experience from the development team of this project BKHack, it was decided that Reason is the language for front-end UI development.

4.1.2. Front-end hosting

For our system, the front-end is a static-site web application i.e. browser requests at a route and always deterministically receives a HTML+JS+CSS bundle. This is contrary to dynamic web applications e.g. web apps traditionally powered by technologies like PHP, where the server is non-trivially program to handle requests at each route. The static-site pattern greatly reduces development complexity for little runtime cost (insofar as non-functional requirements of our system are concerned).

Figure 45: firebase.google.com

Figure 46: netlify.com

4.1.3. Conclusion

OCaml and Reason and their ecosystem of languages and frameworks are known to facilitate documentation which is what we want. This will become relevant in Section 4.3. Reason's support for functional programming and reactive programming through React interop also proves itself to be an attractive choice. That's why we've chosen Reason as the primary programming for developing BKHack's front-end.

4.2. Design system

While established design systems like Material Design [21], Ant Design [22], Apple Human Interface [23], and GNOME HIG [24] provide comprehensive frameworks for general-purpose applications, they prioritize consumer-friendly aesthetics and mobile-first approaches that conflict with our target audience's needs. These systems emphasize visual polish through shadows, animations, and generous whitespace—patterns designed for casual users navigating touch interfaces.

As stated previously in Section 3.9, our technical audience expects the efficiency and information density of developer tools. We determined that adapting an existing consumer-oriented system would require removing more features than we retain, effectively fighting against the design language rather than leveraging it. Therefore, we introduce **the BKHack design system** as a purpose-built collection of rules and reusable components tailored specifically for academic knowledge workers who value functional density over popular aesthetics.[1]

Through multiple iterations, we adopted a terminal-UI aesthetic, and as such, the BKHack design system establishes a cohesive foundation that bridges the gap between a modern web application and the functional directness of terminal-UI (TUI) applications.

4.2.1. Visual hierarchy

Through experimentation, we initially adopted a skeuomorphic design pattern, not too dissimilar to existing platforms like reddit.

Skeuomorphic design[2][3] is a commonly-used UI style and was quite popular in the 2000s. Skeuomorphic design patterns are, for example: drop shadows, gradients, and pseudo-3D effects; originated in an era when digital interfaces needed to mimic physical objects to aid user understanding. For our technical audience, these

[1]Disclaimer: The following subsections outline the visual and structural highlights of the BKHack design system. The full technical specification is maintained in a separate file, serving as the primary reference for implementation alignment, accessibility compliance, and semantic token mapping

[2]https://www.interaction-design.org/literature/topics/skeuomorphism

[3]https://en.wikipedia.org/wiki/Skeuomorph#Arguments_in_favor

affordances are not only unnecessary but actively counterproductive. Shadows consume visual bandwidth without conveying information, gradients introduce color variations that compete with semantic signals, and depth effects conflict with the flat, high-contrast environments (IDEs, terminals, documentation sites) our users inhabit daily. By rejecting skeuomorphism in favor of a flat, surface-layered hierarchy, we achieve several goals: first, we reduce visual noise, allowing content and structure to dominate the interface; second, we increase information density by eliminating decorative padding and effects; third, we create a cohesive aesthetic language that feels native to the development environment rather than borrowed from consumer applications.

To avoid "skeuomorphic" shadows and maintain a flat, technical aesthetic, we implemented hierarchy through surface layering. This system utilizes three distinct tiers:

- The Base (main page background)
- The Mantle (primary containers like cards and navigation)
- The Crust (inputs and insets), which creates an illusion of depth for interactive fields.

We implemented a CLI-inspired header motif for page navigation, such as displaying

```
$ git pr --list | grep open
```

which provides both a literal command-like path and high-level metrics for the current view (e.g., "5 open pull requests"). This motif serves two purposes: it rewards users familiar with terminal commands by providing recognizable syntax, and it subtly incentivizes less experienced users to explore command-line tools by presenting them in a non-threatening reading context.

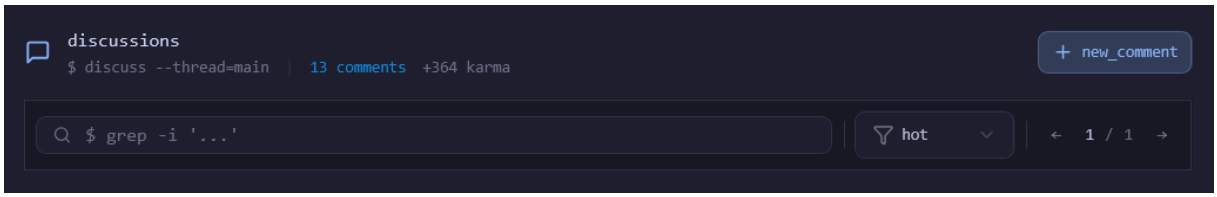

Figure 47: The header of discussions in BKHack

4.2.2. Color palette and semantic meanings

A primary challenge was achieving a visual hierarchy that accommodates high density without causing visual fatigue.

We decided against aggressive color variations in favor of the Catppuccin pastel palette, using the "Mocha" and "Latte" variants to ensure eye strain is reduced during extended use while maintaining dual-theme parity with identical contrast ratios. To reconcile our university identity with this aesthetic, we integrated HCMUT-brand blue (#1488DB) as the primary action color across both themes. We enforced a strict rule that color is never used for mere decoration; instead, it carries consistent semantic meaning:

- Green for success/verification

- Red for danger/destruction
- Mauve for modifications or in-progress states.

Figure 48: The logo of catppuccin

Figure 49: The logo of HCMUT

Figure 50: Catppuccin latte theme

Figure 51: Catppuccin mocha theme

This semantic consistency is critical for our multi-faceted navigation; it ensures that as a user moves between different tabs (e.g., from an Article to an Issue), they do not have to re-learn button functions and can instead rely on muscle memory.

We further implemented a separation of tab signature colors from semantic action colors - while specific sections like “Issues” may have distinct red-adjacent pastel signatures for identification, primary actions within those sections remain “Brand Blue” to prevent users from mistaking a “New Issue” button for a destructive action.

Feedback is provided through status badges using a “redundant encoding” approach, combining color, icons, and text so that “power users” can recognize the state of a post—such as whether it is verified or outdated at a single glance.

4.2.3. Use of typeface

To reinforce the platform’s CS-centric identity, we adopted a two-typefaced system that serves as a structural signifier.

While sans-serif is used for long-form prose to maintain readability, monospaced fonts are strictly applied to headings, timestamps, and metadata labels. This visually distinguishes content from syntax and makes the platform feel like an extension of the developer’s local environment.

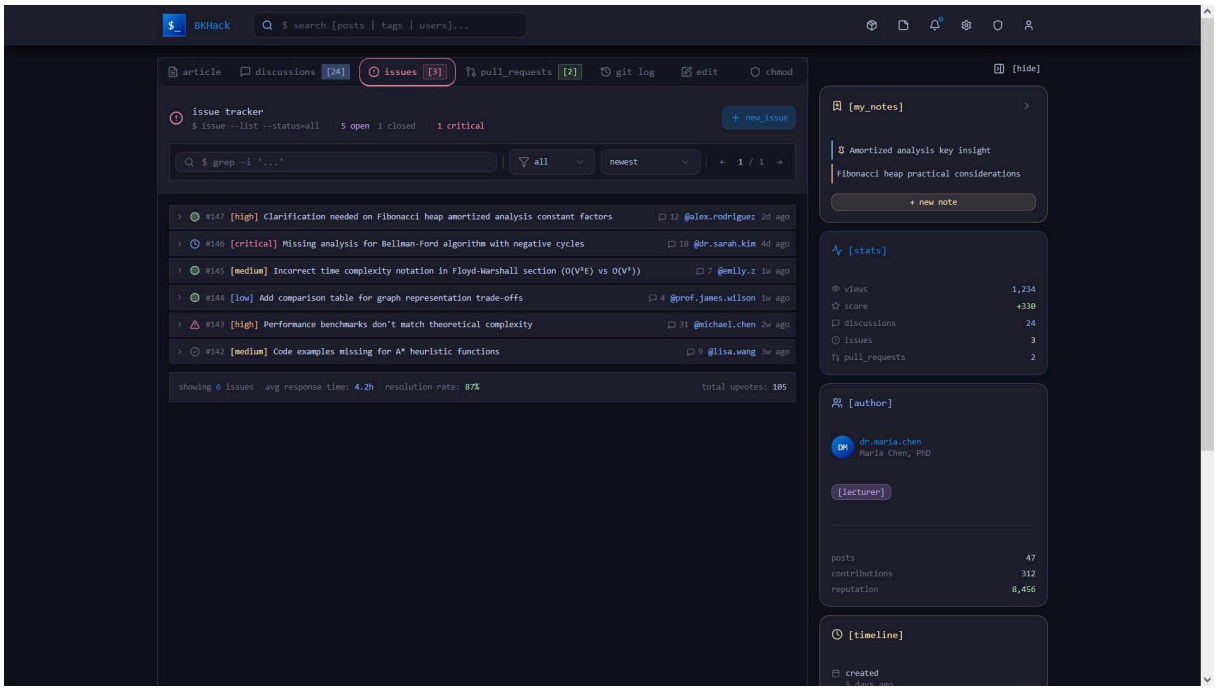

Figure 52: The post issues page of BKHack

4.2.4. Motion and animation

To maintain the platform's directness and respect user preferences, we adhere to a minimal-motion philosophy. All transitions are restricted to essential state changes such as: hover effects (opacity/color shifts ≤200ms), focus indicators, and page transitions. Avoiding decorative animations like slides, bounces, or parallax effects.

This serves in improving perceived performance by eliminating animation delays; it respects users with vestibular disorders or motion sensitivity (consistent with WCAG 2.1 success criterion 2.3.3); and it maintains consistency with the immediate feedback paradigm of terminal applications. The system automatically respects the `prefers-reduced-motion` media query, disabling even essential transitions for users who request it at the OS level.

4.2.5. Visual vocabulary and text-based graphics

Rather than relying on custom illustrations or icon-heavy navigation, we prioritize text-based visual elements but not entirely omitting icons. Status indicators combine text labels, icons, and color in a redundant encoding pattern, ensuring accessibility while allowing power users to scan states at a glance (e.g., a "Verified" badge shows green color + checkmark icon + "Verified" label simultaneously).

By taking a text-first approach it reduces dependency on visual assets, improves accessibility for screen readers and high-contrast modes.

4.3. Documentation rituals

During this project, every member of the development team is encouraged to document their work and share their ideas as much as possible. We believe it is a way to boost morale and bolster communication.

Write manuals / guides MD files for developers on how to get certain things done. Can be opened and read publicly. Write MDX files. Can be opened with Storybook.

Record design decisions.

Write interface files. Follow the official ocaml guide [25].

- Write documentation in .mli files
- Write introductory documentation for the toplevel module
- Organize signatures into logical sections
- Document all your signatures
- Put documentation comments after signature elements
- Write usage examples
- Properly document deprecations
- Have meaningful README

4.4. Layering of styles

We consider it in development time and also in run time that styles should be implemented in a specific order. This order is so that we can achieve the ideas presented in Section 4.2.4 and Section 4.2.5.

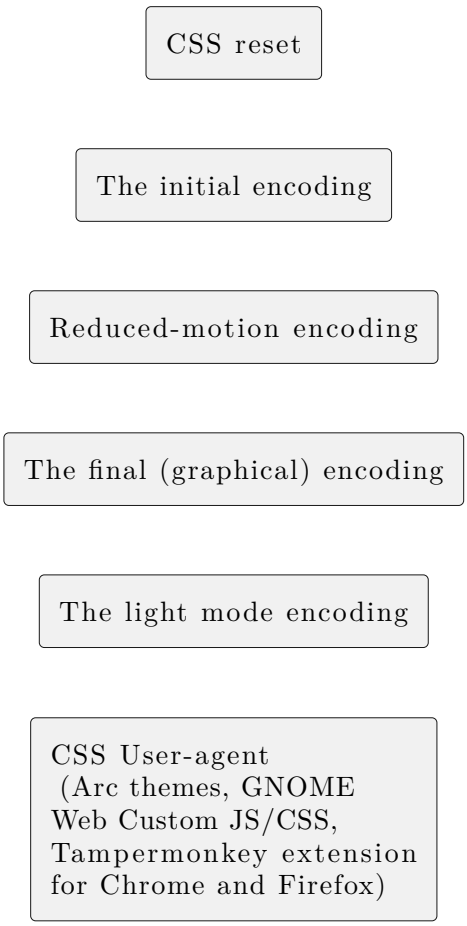

Figure 53: The ordered layers of styling

4.5. Bundling of front-end

Web applications adopt one of these paradigms:

- Multi-page application (MPA)
- Single-page application (SPA)

BKHack adopts a hybrid paradigm. At its core, it's MPA. But within a webpage there can be some "subpages", "subviews", which don't require a request to server to navigate to.

All static asset - html document, css stylesheet, icon svg - should be separated into files. This is opposite to the inline philosophy e.g. inline styling, css-in-js, etc.

4.6. Source code

The implementation source code can be found at

https://github.com/ttb-hcmut/bkhack

which is hosted on the GitHub platform, and where there will be a detailed documentation greeting file (a.k.a. a `README.md` file) to provide quick guide for internal developer users who may use our system as foundation to extend with more pages and services. Figure 54 shows a section in this `README.md`.

Figure 54: A section of the front page of the source code repository of the bkhack system as seen on github.com. We provide an installation flow where BKHack (here writen as bkhack) is easily usable as an OCaml / Reason library through a few short OPAM installation commands. Source: window capture at https://github.com/ttb-hcmut/bkhack

5. Deployment and testing

5.1. Deployment

We support a diverse selection of deployment platforms. To achieve this, we designed it so that our system, once programmed in terms of source code, can be compiled and bundled with assets to produce bundles of deployment artifacts. The details of these artifacts are discussed in Section 5.1.1.

5.1.1. Deployment artifacts

The artifacts of our system can be described by Figure 55. This can be generated from the `bkhack.bundle` command of our system. Developer of our system can use these bundles to deploy to their service of choice.

- For the front-end bundle, the developer opens a shell in the bundle folder, then run the service-specific deployment command. For Firebase Hosting, it is the `firebase deploy` command.
- For the back-end bundle, the developer opens a shell in the bundle folder, then run the service-specific deployment command. For Fly.io, it is the `fly` or `flyctl` command.

For some services, there are specialized CLI commands to manage deployment e.g. Fly.io, and a configuration file is recommended (as is the case of Render.com) or required (as is the case of Fly.io). This is why each bundle includes optional service-specific configuration files with quality-of-life configurations so that the developer experience is as smooth as possible.

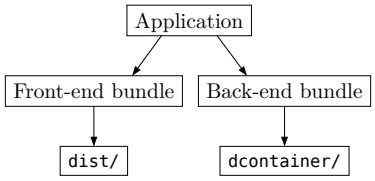

Figure 55: The BKHack system as viewed by deployable artifacts. They are two bundles: the front-end bundle, and the back-end bundle.

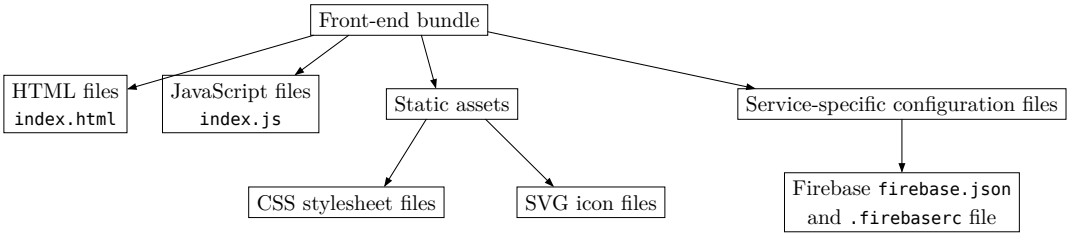

Figure 56: The front-end bundle of the BKHack system, continued from Figure 55.

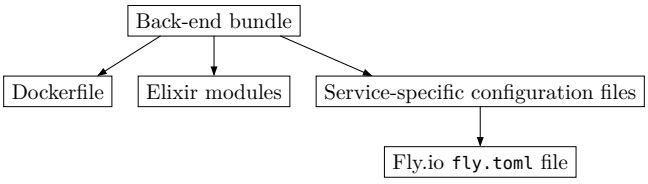

Figure 57: The back-end bundle of the BKHack system, continued from Figure 55.

5.2. Unit testing

For our system, there are certain functions and modules that are more complex than the rest.

5.2.1. Shell parser

Because we implemented a shell parser instead of reusing external libraries, there is a need to test this parser. Unit tests for the shell parser is implemented in the form of inline tests.

5.2.2. Diffing

Because we implemented a diffing algorithm besides reusing ones in the form of external libraries, there is a need to test this algorithm. We've prepared a suite of tests which can be run at anything using the command `dune exec test/diff.exe`.

6. Conclusion

6.1. Achievement

Throughout the initial phase of this project, we have successfully completed the planning, analysis, and high-level specification required to build BKHack.

We have determined the identity and critical functions our implementation has to achieve through analysis of existing systems, conducted surveys, and current affairs.

We have finalized a robust architectural design utilizing a Service-Oriented Architecture (SOA), which provides the optimal balance of modularity, cost-effectiveness, and data integrity that we set as our core attributes. Furthermore, we have defined a comprehensive set of functional requirements and rigorous non-functional requirements regarding performance, security, and accessibility.

Our design phase culminated in a stylish and practical aesthetic, which ensures high information density and is tailored to the specific needs of our demographic - computer science students and lecturers.

6.2. Work-in-progress

With the specification phase complete, the project has transitioned into implementation. While we successfully visualized a prototype to validate visual and structural rules, we are only making partial progress towards building the final product using ReasonML to leverage a more robust, formally-proven functional programming ecosystem. Currently, development is focused on Phase One of the implementation timeline, which includes the authentication system, user profile management, and the core article lifecycle.

We are also establishing our documentation procedures and standards, ensuring that all production code is accompanied by a corresponding documentation and architectural records to maintain long-term project sustainability.

6.3. To-do

The next phase of development will focus on the more complex social and collaborative features of the platform. Key upcoming milestones include:

- Implementation of the Pull Request and Issue Systems: Developing the interfaces and logic for community-sourced fact-checking and structured content refinement.
- Discussion and Note Modules: Building the threaded discussion components that allow for version-specific referencing and personal knowledge management.
- Advanced Moderation Tools: Creating the administrative suite for managing tags, taxonomies, and user roles.
- Deployment and Testing: Setting up the automated build pipeline and conducting full-scale performance and security testing on campus-hosted infrastructure.

Development will continue according to our established schedule until the product operates at its full capacity.

Bibliography

[1] "Hacker News." [Online]. Available at: https://news.ycombinator.com/

[2] "Lobsters." [Online]. Available at: https://lobste.rs/

[3] "Reddit." [Online]. Available at: https://reddit.com/

[4] "X (formerly Twitter)." [Online]. Available at: https://x.com/

[5] [Online]. Available at: https://wiki.gg/

[6] "The Optics of Language-Integrated Query."

[7] "Finally, Safely-Extensible and Efficient Language-Integrated Query."

[8] "What is SOA?," in *The Open Group SOA Source Book*, Van Haren, 2009.

[9] Paul Clements et al, *Documenting Software Architectures 2nd Edition*. Pearson Education, 2010.

[10] "Documenting an architecture is a matter of documenting the relevant views and then adding documentation that applies to more than one view.," in *Documenting Software Architectures 2nd Edition*, Pearson Education, 2010, p. 22.

[11] "Reactive programming." [Online]. Available at: https://en.wikipedia.org/wiki/Reactive_programming

[12] "The Elm Architecture." [Online]. Available at: https://guide.elm-lang.org/architecture/

[13] "Immediate mode (computer graphics)." [Online]. Available at: https://en.wikipedia.org/wiki/Immediate_mode_(computer_graphics)

[14] "Reason • Reason lets you write simple, fast and quality type safe code while leveraging both the JavaScript & OCaml ecosystems.." [Online]. Available at: https://reasonml.github.io/

[15] "Objective ML: An Effective Object-Oriented Extension to ML." doi: 10.1145/263699.263707.

[16] "React - the library for web and native user interfaces." [Online]. Available at: https://react.dev/

[17] "ReasonReact • All your ReactJS knowledge, codified.." [Online]. Available at: https://reasonml.github.io/reason-react/

[18] "Elm - delightful language for reliable web applications." [Online]. Available at: https://elm-lang.org/

[19] "A gentle introduction to Haskell." doi: 10.1145/130697.130698.

[20] "The Elm Architecture - An Introduction to Elm." [Online]. Available at: https://guide.elm-lang.org/architecture/

[21] "Material Design." [Online]. Available at: https://m3.material.io/

[22] "Ant Design." [Online]. Available at: https://ant.design/docs/spec/introduce/

[23] "Apple Human Interface Guidelines." [Online]. Available at: https://developer.apple.com/design/human-interface-guidelines

[24] "GNOME Human Interface Guidelines." [Online]. Available at: https://developer.gnome.org/hig/

[25] "OCaml Documentation Guidelines." [Online]. Available at: https://ocamlverse.net/content/documentation_guidelines.html

[26] "Shell Command Language." [Online]. Available at: https://pubs.opengroup.org/onlinepubs/9699919799/utilities/V3_chap02.html

[27] "Stream-based programming, as compared to sh." [Online]. Available at: https://github.com/ttb-hcmut/bkhack/tree/main/src_devbook/streaming.pdf

[28] "Chapter 5.4 Work Assignment Style.," in *Documenting Software Architectures 2nd Edition*, Pearson Education, 2010, p. 202.

[29] Phan Thị Tươi, *Giáo trình Trình biên dịch*. Nhà xuất bản Đại học Quốc gia Thành phố Hồ Chí Minh.

[30] "Diffing." [Online]. Available at: https://github.com/ttb-hcmut/bkhack/tree/main/src_devbook/diff.pdf

[31] Eugene W. Myers, "An O(ND) Difference Algorithm and Its Variations."

Appendix

A. Logo

Figure 58: The full logo of BKHack

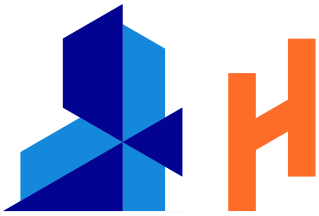

Figure 59: The compact logo of BKHack

The BKHack logo is explicitly grounded in the official visual identity of Ho Chi Minh City University of Technology (HCMUT), and its design choices closely follow the university's brand recognition principles while extending them to fit its own philosophy.

According to HCMUT's brand documentation[4], the university logo is conceived as a three-dimensional spatial form, centered on a regular hexagon inscribed within a circle. Its structure is inspired by honeycomb architecture, a form regarded as both simple and scientific. This geometry symbolically represents diligence, solidity, and creativity—values strongly associated with HCMUT's academic culture. The color system is built from two shades of blue, conveying calmness, rigor, and formality.

BKHack preserves this conceptual foundation. The hexagonal outline remains the dominant structural element, maintaining immediate visual continuity with the HCMUT logo. Rather than treating the hexagon as a mere container, the BKHack logo integrates it directly with the "BK" letterform. The outline and the letters visually interlock, making the boundary of the logo feel structural rather than decorative. This blending mirrors software concepts where interfaces, abstractions, and implementations are tightly coupled, reinforcing the idea that BKHack is an internal, purpose-built system rather than an external add-on.

In contrast to the strictly institutional palette, the inclusion of orange in the "Hack" portion of the logotype introduces a deliberate deviation. This color choice serves as a subtle nod to Hacker News, a well-known social news platform within the global computer science and software engineering community.

Taken together, the BKHack logo inherits HCMUT's geometric rigor, symbolism, and color discipline that is recognizably related to the university, yet clearly positioned as a distinct platform.

[4]https://hcmut.edu.vn/gioi-thieu/nhan-dien-thuong-hieu

B. Programming language parsing

Markdown. For the writing experience of our website, we've used and tests a collection of publicly-available markdown parsers in both the JavaScript ecosystem and the Reason / OCaml ecosystem.

Shell language. For the command bar of our website, we've developed a robust command language based on the Shell Command Language [26]. A more detailed discussion can be found in our developer handbook at [27].