DESIGN AND DEVELOPMENT OF A CAMPUS-BASED DIGITAL COMMUNITY PLATFORM FOR ACADEMIC INTERACTION AND KNOWLEDGE SHARING IN NIGERIAN UNIVERSITIES

BY

ERHARHINE OGHENEKEVWE JOSHUA

PSC2105330

DEPARTMENT OF COMPUTER SCIENCE,
FACULTY OF COMPUTING,
UNIVERSITY OF BENIN,
BENIN CITY,
EDO STATE NIGERIA

NOVEMBER 2025

DESIGN AND DEVELOPMENT OF A CAMPUS-BASED DIGITAL COMMUNITY PLATFORM FOR ACADEMIC INTERACTION AND KNOWLEDGE SHARING IN NIGERIAN UNIVERSITIES

BY

ERHARHINE OGHENEKEVWE JOSHUA

PSC2105330

A PROJECT REPORT SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE, FACULTY OF COMPUTING, UNIVERSITY OF BENIN, BENIN CITY, IN PARTIAL FULFILMENT OF THE REQUIREMENT FOR THE AWARD OF A BACHELOR OF SCIENCE (B.Sc.) DEGREE IN COMPUTER SCIENCE

NOVEMBER 2025

CERTIFICATION

This is to certify that this project work was carried out by ERHARHINE OGHENEKEVWE JOSHUA with Matriculation Number PSC2105330 under my supervision. It is adequate and satisfactory, both in scope and content, for the award of Bachelor of Science (B.sc) Degree in Computer Science of the University of Benin

Mr. K.O. Otokiti **DATE**

Project Supervisor

APPROVAL

This project work is hereby approved in partial fulfilment of the requirements for the award of Bachelor of Science (B.Sc.) Degree in Computer Science from the University of Benin.

Dr. (Mrs.) A.R. Usiobaifo **DATE**

Ag Head of Department

DEDICATION

This project is dedicated to God Almighty for giving me the strength and wisdom to see it through to completion, and even throughout my stay in the University of Benin (UNIBEN). It is also dedicated to my parents; Mr and Mrs Paul and Ifeoma Erharhine, OgheneTega Erharhine, Ufuoma Erharhine, and Esseoghene Erharhine for their love, sacrifice, support and guidance throughout my academic journey.

ACKNOWLEDGEMENT

My utmost acknowledgement goes to God Almighty for giving me the strength, wisdom and direction throughout my academic journey. I would like to express my gratitude to my project supervisor Mr. K.O. Otokiti for his consistent guidance towards ensuring the successful completion of this project.

I would also like to specially thank my project coordinator Dr. Osagi Maxwell, and other lecturers in the Department of Computer Science who I have been opportune to cross paths with, and have impacted me immensely these past few years: Prof. F.I. Amadin, Dr. E. Nweli, Prof. G.O. Ekuobase, Dr. F.O. Oliha, Prof. (Mrs.) V.V.N. Akwukwuma, , Prof. (Mrs.) S. Konyeha, Prof. (Mrs.) V.I. Osubor, Dr. (Mrs.) Aziken, Prof. F.O. Chete, Dr. (Mrs) R.O. Osaseri, Dr. J.C. Obi, Mr. P. E.B. Imiefoh, Mr. I.E. Obasohan, Mr. K.O. Otokiti, Mr. I.E. Obayagbonna, Mrs. R.I. Izevbizua, Mr. E.C. Igodan, Miss L.O.Usiosefe, Mr J. Okhuoya, and Mr. D.N. Idehen.

I would also like to thank my family and friends for their support, words of encouragement, and consistent guidance throughout this project.

TABLE OF CONTENTS

LIST OF TABLE

ABSTRACT

This project presents the design and development of **BenTalk**, a centralized digital community platform specifically tailored for academic interaction and knowledge sharing within Nigerian universities, with a prototype implementation for the University of Benin's Faculty of Computing. The study addresses the critical challenge of fragmented academic communication, where students currently rely on general-purpose social media platforms like WhatsApp, Facebook, and Telegram—platforms fundamentally designed for social interaction rather than structured academic discourse.

Through comprehensive literature review and system analysis, the research identifies key limitations of existing communication channels: information fragmentation across multiple platforms, lack of institutional oversight, absence of knowledge preservation mechanisms, erosion of professional boundaries, and algorithm misalignment with educational objectives. These challenges necessitate a purpose-built solution that balances the accessibility of social media with the structure required for effective academic collaboration.

The BenTalk platform employs a three-tier architecture consisting of a Python FastAPI backend with PostgreSQL database, a React-based responsive web application, and a native Android mobile application developed using Kotlin and Jetpack Compose. The system implements a hierarchical subspace structure organized by departments (Computer Science, Cyber Security, Data Science, Software Engineering, Information and Communication Technology, Information Technology, and Information Science) and academic levels (100L, 200L, 300L, 400L), aligning with the university's existing organizational framework.

Core functionalities include secure user authentication via JWT tokens, threaded discussion forums with nested commenting capabilities, upvote/downvote mechanisms for content quality signaling, full-text search across posts, and real-time updates through WebSocket integration. The platform emphasizes knowledge preservation through permanent, searchable archives that benefit future student cohorts while maintaining intuitive navigation and mobile-first design principles suited to Nigerian infrastructure constraints.

The prototype demonstrates technical feasibility and addresses identified gaps in current academic communication systems. Testing confirms that the platform successfully provides structured academic discourse spaces, reduces information redundancy, facilitates peer-to-peer learning, and enables optional institutional oversight without compromising student ownership of discussions. The system's modular architecture allows for scalability and adaptation to other faculties and institutions.

This research contributes to the growing body of literature on educational technology in African contexts by demonstrating that locally-developed, context-appropriate solutions can effectively address challenges that generic global platforms cannot. The project provides a blueprint for similar implementations across Nigerian universities and offers practical recommendations for institutional adoption, technical enhancement, and sustainable deployment.

Keywords: Academic collaboration, discussion platform, knowledge sharing, Nigerian universities, educational technology, social media in education, FastAPI, React, mobile application, University of Benin

CHAPTER ONE
INTRODUCTION

1.1 Background of the Study

The gradual commercialization of electronic devices over the past two decades has metamorphosed the way people communicate, amassing large communities where students can share their work publicly, engage with specialists in their discipline, and contribute to the parity of educational opportunity (Macht et al., 2020). This digital transformation has fundamentally altered the landscape of higher education, creating new possibilities for collaboration and knowledge dissemination.

Specifically in more recent years, social media has become an integral part of formal education in higher institutions, particularly in the post-COVID-19 era. Platforms such as Facebook, YouTube, WhatsApp, Twitter, and Instagram demonstrate the greatest dominance in pedagogical aid among educators who have adopted social media technologies. Students have been observed to use these platforms extensively for academic discourse. However, this fragmentation reveals that social media was not originally intended for academic use and is currently being improvised for educational purposes (Aqdas et al., 2020).

In Nigerian universities, students face additional challenges including overcrowded classrooms, limited access to academic materials, and insufficient platforms for peer learning (Adewale & Aremu, 2020). The social media platforms currently relied upon are largely set up by students themselves, creating a critical disconnect between students and the institution. While some lecturers attempt to fill this void, they are often required to provide personal contact information, which unfortunately blurs the line between social interactions and the classic formal setting of a higher institution (Macht et al., 2020). There is, therefore, a pressing need for a dedicated,

university-focused digital platform that combines the structure of global knowledge-sharing systems with the academic focus required in higher education.

Globally, platforms like Usenet, Quora, and Reddit (launched 2005) have demonstrated how structured online forums can foster meaningful discourse. Quora, for instance, provides a question-and-answer model where users post queries and receive responses from peers or experts. Reddit, on the other hand, organizes discussions into subreddits, which serve as sub-communities focused on particular topics. These models highlight the value of structured, searchable, and community-driven discussions (Zhao & Rosson, 2009). However, their global and generalized nature makes them less suitable for the specific needs of academic institutions.

This project seeks to address that gap by designing and implementing a discussion forum application specifically for a university setting. The system is intended to provide the Faculty of Computing and its departments with dedicated subspaces where students can post questions, share answers, and engage in threaded discussions. Unlike generic platforms, the proposed solution **BenTalk** is tailored to the academic environment, emphasizing knowledge preservation, accessibility, and peer-to-peer collaboration.

Technologically, the project leverages modern frameworks to ensure cross-platform accessibility. The backend is implemented using FastAPI (a high-performance Python framework), with PostgreSQL for persistent data storage and SQLAlchemy as the Object-Relational Mapper (ORM). The frontend consists of a React-based web application and a Kotlin-based Android mobile app, ensuring that both desktop and mobile users can access the platform seamlessly.

By offering an organized, structured, and academic-focused hub, this system provides students with a reliable space to collaborate, reduces reliance on scattered chat groups, and encourages knowledge-sharing communities within universities. The platform is designed specifically for the

University of Benin's Faculty of Computing, with a hierarchical structure that accommodates different departments (Computer Science, Cyber Security, Data Science, Software Engineering, Information and Communication Technology, Information Technology, and Information Science) and level-specific discussion spaces (100, 200, 300, and 400 levels).

1.2 Statement of the Problem

University students currently face significant limitations and vicarious challenges in the adoption of social media networks for academic utilization. These challenges can be categorized as follows:

Fragmentation of Information: Essential academic topics are spread across multiple platforms, often buried under endless extraneous data such as jokes, spam, and personal conversations. Students struggle to locate relevant information when it is needed, leading to duplicated questions and inefficient knowledge sharing.

Absence of Institutional Management: The lack of proper moderation and security oversight allows misinformation and impersonation to flourish. Without institutional control, there is no mechanism to verify the authenticity of information or the identity of users sharing academic content (Van Den Beemt et al., 2019).

Lack of Structure: Social media platforms are fundamentally designed to connect people on a personal rather than professional basis (Aqdas et al., 2022; Macht et al., 2020). This leads to conversations that mix academic content with social banter, making it difficult to maintain focus on educational objectives.

Erosion of Boundaries: The informal nature of social media inadvertently leads to the erosion of professional boundaries. Sensitive data such as personal contact information becomes exigent

(Macht et al., 2020), and the distinction between personal and professional communication becomes blurred (Van Den Beemt et al., 2019).

Algorithm Misalignment: The algorithms behind average social media platforms are designed to maximize social engagement and advertising revenue, not academic discourse. This means that the most engaging (but not necessarily most educational) content rises to the top, while important academic announcements may be missed entirely.

Information Deluge: Students are overwhelmed by the sheer volume of content on social media platforms, making it difficult to filter relevant academic information from the noise (Macht et al., 2020).

Lack of Knowledge Preservation: Unlike traditional learning management systems, social media platforms do not preserve academic discussions in a searchable, organized manner. Important questions and answers are lost in endless chat histories, requiring students to repeatedly ask the same questions.

Privacy and Security Concerns: Students and faculty have legitimate concerns about privacy when using public social media platforms for academic purposes. There is no control over who can access discussions or how personal information might be used.

These limitations highlight the urgent need for a dedicated academic discussion platform that can organize knowledge, foster collaboration, ensure institutional oversight, and maintain appropriate professional boundaries all while remaining accessible to students within the University of Benin community.

1.3 Aim and Objectives

Aim

The aim of this project is to develop **BenTalk**, a centralized discussion platform specifically designed for the University of Benin's Faculty of Computing, attempting to realign with the

university's philosophical framework of structured academic discourse while providing students with a dedicated space for knowledge sharing and collaboration.

Objectives

To achieve this aim, the project pursues the following specific objectives:

1. To design a centralized platform where departments of the Faculty of Computing can create and manage dedicated subspaces for level-specific discussions (100L, 200L, 300L, and 400L).
2. To enable students to post questions, provide answers, and engage in threaded discussions within their respective departmental and level-specific contexts.
3. To incorporate engagement features such as upvotes and comments to improve content quality and help surface the most useful contributions from the community.
4. To build a responsive web interface using React that provides an intuitive user experience across desktop and mobile browsers.
5. To develop a native Android mobile application using Kotlin that offers full platform functionality for mobile users.
6. To implement a robust backend system using FastAPI and PostgreSQL that handles user authentication, data persistence, and real-time updates via WebSocket connections.
7. To ensure scalability, accessibility, and ease of use across various user groups within the university, including students across different academic levels and departments.
8. To create a knowledge preservation system where academic discussions are permanently archived and searchable, preventing the loss of valuable information.

9. To develop a prototype that can serve as proof-of-concept for potential deployment across other faculties and universities in Nigeria.

1.4 Scope of the Study

This project is focused on the prototype development of an academic discussion platform with specific boundaries and limitations:

What is Included:

Target Users: Students and lecturers within the Faculty of Computing at the University of Benin, Nigeria.

Departments Covered:

- Computer Science (CSC)
- Cyber Security (CYB)
- Data Science (DSC)
- Software Engineering (SFE)
- Information and Communication Technology (ICT)
- Information Technology (IT)
- Information Science (IS)

Academic Levels: 100 Level, 200 Level, 300 Level, and 400 Level discussions for each department.

Core Features:

- User registration and authentication
- Department and level-based subspace navigation
- Post creation with title and content
- Commenting system with nested replies

- Upvote/downvote functionality for posts
- Real-time updates via WebSocket
- Search functionality for posts
- User profiles with academic information

Platforms:

- Web application (React-based, cross-browser compatible)
- Android mobile application (Kotlin-based)

Technology Stack:

- Backend: Python FastAPI, PostgreSQL database, SQLAlchemy ORM
- Frontend Web: React, Dexie.js for local storage
- Frontend Mobile: Kotlin, Jetpack Compose, Retrofit for API communication

Testing: Prototype usability and functionality testing with a small group of students and potential heuristic evaluation by experts.

What is NOT Included:

Integration Limitations: The system does not integrate with existing university portals, student information systems, or official university databases.

Advanced Features Not Implemented:

- AI-driven content moderation or recommendation systems
- Automated plagiarism detection
- Video/audio conferencing capabilities
- Gamification elements (badges, points, leaderboards)
- File attachment beyond basic text and links
- Advanced analytics and reporting dashboards

Scope Limitations:

- No faculty-wide deployment or live production environment
- No integration with course management systems (e.g., Moodle)
- No official institutional approval or adoption
- Limited to Faculty of Computing (not university-wide)

Security Features Not Fully Implemented:

- Advanced encryption for sensitive data
- Two-factor authentication
- Comprehensive audit logging
- GDPR-compliant data management

Scale Limitations: The prototype is designed for testing and demonstration purposes, not for handling thousands of concurrent users in a production environment.

1.5 Research Methodology

This research adopts a **Development-Focused Research Methodology** to design, develop, and preliminarily evaluate the proposed centralized university discourse application. Given that the project is in its development phase without extensive field testing, the research approach is qualitative and analytical, relying on expert feedback and structured evaluation against predefined criteria to assess the project's feasibility, usability, and potential impact.

1. Research Philosophy: Pragmatism

The research is guided by a pragmatist philosophy, which is concerned with what works best in solving practical problems. The focus is on the practical outcome of building a functional application and evaluating its potential utility in a real-world university context, rather than

seeking a single, objective truth. This approach acknowledges that the value of the system lies in its practical application and its ability to solve identified problems in the academic environment.

2. Research Design: Descriptive Case Study Design

The project is treated as a descriptive case study in the development of a software solution for a specific problem (fragmented university communication). The design involves a detailed, in-depth description of the application's architecture, features, and evaluation against stated objectives. This approach allows for comprehensive documentation of the development process and the rationale behind design decisions.

3. Data Collection Methods

In the absence of large-scale field testing, data is collected through:

Architectural and Code Documentation: The technical design, including the Python-based backend (using FastAPI) and the Kotlin-based Android client, serves as a primary artifact for evaluation. This includes:

- System Architecture Diagrams (Client-Server model, REST API design)
- Database Schema (PostgreSQL entity-relationship diagrams)
- Source code demonstrating implementation of key features
- API documentation and endpoint specifications

1.6 Significance of the Study

This project holds significant value for multiple stakeholders in the academic ecosystem:

For Students

Knowledge Preservation: Unlike ephemeral WhatsApp messages or buried Facebook posts, BenTalk provides a permanent, searchable archive of academic discussions. Students can

reference past questions and answers, reducing redundant inquiries and building a collective knowledge base.

Peer-to-Peer Learning: The platform encourages collaboration, problem-solving, and mentoring among students across different levels. Senior students can share insights and experiences with juniors, fostering a culture of mutual support.

Focused Academic Discourse: By providing a dedicated academic space, students can engage in scholarly discussions without the distractions and noise of general social media platforms.

Accessibility: With both web and mobile interfaces, students can access the platform from anywhere, at any time, using devices they already own.

For Faculty and Lecturers

Institutional Oversight: Faculty members can monitor discussions, provide authoritative answers, share important announcements, and ensure information accuracy within their departments.

Student Engagement Insights: The platform provides faculty with visibility into the questions and challenges students face, helping them adjust teaching methods and course content accordingly.

Reduced Communication Overhead: Instead of answering the same questions repeatedly via email or WhatsApp, lecturers can provide comprehensive answers once on the platform, which remain accessible to all students.

For the Institution

Enhanced Communication Infrastructure: The university gains a structured communication channel that bridges the gap between formal announcements and informal student interactions.

Quality Assurance: With moderation capabilities, the institution can ensure that information shared on the platform is accurate and appropriate.

Community Building: The platform fosters a sense of community within the Faculty of Computing, potentially improving student retention and satisfaction.

Data for Decision Making: Aggregated (anonymized) data about common student queries and concerns can inform policy decisions and curriculum improvements.

For Academic Research

Technological Advancement: The project demonstrates the application of modern software development frameworks (FastAPI, React, Kotlin, PostgreSQL) in solving real-world educational problems.

Contribution to Literature: This work adds to the growing body of research on the role of technology in higher education, specifically addressing the Nigerian context where infrastructure challenges and resource constraints are significant factors.

Replicability: The open-source nature of the core technologies means other institutions can adapt and deploy similar solutions.

For Nigerian Higher Education

Scalability: This prototype serves as a model that can be scaled for broader use across multiple universities in Nigeria and beyond.

Cost-Effectiveness: By using open-source technologies and modern development practices, the platform can be deployed and maintained at minimal cost compared to proprietary solutions.

Addressing Local Challenges: The platform is specifically designed with the constraints of Nigerian universities in mind, including limited internet bandwidth, varied device capabilities, and the need for offline functionality where possible.

Digital Transformation: The project contributes to the broader digital transformation efforts in Nigerian education, demonstrating how locally-developed solutions can address contextual challenges.

1.7 Definition of Terms

Academic Discourse: Structured conversation and debate about scholarly topics, characterized by evidence-based argumentation, critical thinking, and respect for diverse perspectives.

API (Application Programming Interface): A set of protocols and tools that allows different software applications to communicate with each other. In this project, the REST API enables the frontend applications to interact with the backend server.

Authentication: The process of verifying the identity of a user before granting access to the system. This project uses username/password authentication with JWT (JSON Web Tokens).

Backend: The server-side component of the application that handles business logic, database operations, authentication, and data processing. BenTalk's backend is built with Python and FastAPI.

Database: An organized collection of structured information or data, typically stored electronically. This project uses PostgreSQL, a relational database management system.

Dexie.js: A wrapper library for IndexedDB, a JavaScript-based object-oriented database for storing significant amounts of structured data in web browsers.

Discussion Platform: A digital environment where users can post questions, share information, and interact through structured conversations organized by topics or categories.

FastAPI: A modern, high-performance Python web framework for building APIs. It is known for its speed, ease of use, and automatic generation of API documentation.

Frontend: The client-side component of the application that users interact with directly. This includes the visual interface and user experience elements. BenTalk has both web (React) and mobile (Kotlin) frontends.

Jetpack Compose: Android's modern toolkit for building native user interfaces using Kotlin. It simplifies and accelerates UI development with declarative programming.

Kotlin: A modern, statically-typed programming language that runs on the Java Virtual Machine (JVM) and is officially supported for Android app development.

ORM (Object-Relational Mapper): A programming technique that lets you query and manipulate data from a database using an object-oriented paradigm. This project uses SQLAlchemy as its ORM.

PostgreSQL: A powerful, open-source object-relational database system with over 30 years of active development, known for reliability, feature robustness, and performance.

React: A popular JavaScript library for building user interfaces, developed and maintained by Facebook (Meta). It allows developers to create reusable UI components.

REST (Representational State Transfer): An architectural style for designing networked applications. RESTful APIs use HTTP requests to perform CRUD (Create, Read, Update, Delete) operations.

Retrofit: A type-safe HTTP client for Android and Java, used in this project to make network requests from the Android app to the backend API.

SQLAlchemy: A comprehensive SQL toolkit and Object-Relational Mapper for Python that provides a full suite of well-known enterprise-level persistence patterns.

Subspace: A dedicated discussion area within the platform focused on a specific department and academic level (e.g., "Computer Science 200 Level"). Analogous to subreddits on Reddit or channels on Slack.

Threaded Discussion: A conversation structure where replies are linked directly to specific posts, creating a tree-like hierarchy that makes it easy to follow related comments.

Upvote/Downvote: A feature that allows users to express approval or disapproval of content, helping surface high-quality contributions and filter out less useful information.

WebSocket: A communication protocol that provides full-duplex communication channels over a single TCP connection, enabling real-time, bidirectional communication between client and server.

CHAPTER TWO
LITERATURE REVIEW

2.1 Introduction

This chapter provides a comprehensive review of existing literature related to social media use in higher education, the evolution of online discussion platforms, and the specific challenges faced by Nigerian universities in adopting technology for academic purposes. The review establishes the theoretical and practical foundations for the development of BenTalk, identifying gaps in current solutions and justifying the need for a dedicated academic platform.

The chapter is structured to first examine the conceptual framework of social media in education, then trace the historical evolution of online communication platforms, analyze current usage patterns and challenges, review existing systems, and finally synthesize findings to position BenTalk within the broader landscape of educational technology.

2.2 Conceptual Framework

2.2.1 Social Media Defined

Social media is broadly defined as web 2.0 technology that enables collaboration, community building, and opportunity for interpersonal relationships, usually in an informal manner (Tess, 2013). The term "web 2.0" itself refers to the second generation of internet services that emphasize user-generated content, usability, and interoperability. Unlike static websites of the web 1.0 era, social media platforms are dynamic, interactive, and user-centric.

Boyd and Ellison (2007) provide a more specific definition of social networking sites as web-based services that allow individuals to: (1) construct a public or semi-public profile within a bounded system, (2) articulate a list of other users with whom they share a connection, and (3) view and traverse their list of connections and those made by others within the system. This

definition captures the relational nature of social media that distinguishes it from earlier forms of online communication.

2.2.2 Theoretical Foundations

Several theoretical frameworks inform our understanding of how social media can support learning in higher education:

Connectivism Theory: Proposed by Siemens (2005), connectivism positions learning as a process of creating connections and developing networks. In the digital age, the ability to access and synthesize information distributed across multiple sources becomes more important than storing information in one's own memory. Social media platforms facilitate the creation of these learning networks, enabling students to tap into collective knowledge.

Community of Practice Theory: Wenger's (1998) concept of communities of practice describes groups of people who share a concern, set of problems, or passion about a topic, and who deepen their knowledge and expertise by interacting on an ongoing basis. Academic departments and level-based student groups can be understood as communities of practice, and digital platforms like BenTalk provide the infrastructure for these communities to thrive online.

Social Constructivism: Building on Vygotsky's work, social constructivism emphasizes that knowledge is constructed through social interaction. Learners build understanding through dialogue, collaboration, and exposure to multiple perspectives. Social media platforms, with their emphasis on conversation and user-generated content, align well with social constructivist principles.

2.2.3 Parity of Opportunity in Digital Education

Modern technology has greatly reduced the cost of electronic devices while advancing their functionality and features. Social media, with its support for user-generated content, discussion

forums, and web 2.0 features, enables anyone with access to an electronic device and internet connectivity to learn and interact with knowledge from anywhere in the world, regardless of language or culture (Macht et al., 2020).

This democratization of access represents a significant shift in educational possibilities. Students in Nigerian universities, who may face limitations in physical infrastructure and library resources, can potentially access the same information and engage with the same communities as students in well-resourced Western institutions. However, as this review will show, realizing this potential requires addressing specific challenges related to platform design, institutional support, and digital literacy.

2.3 History and Evolution of Social Media in Education

2.3.1 PLATO: The Pioneer (1960s-1980s)

The Programmed Logic for Automatic Teaching Operations (PLATO) system is highly regarded as the pioneer in the use of social media features in a learning management system (Woolley, 2016). Developed at the University of Illinois in 1960 by Professor Don Bitzer at the Computer-Based Education Research Laboratory, PLATO's architecture was built with plasma terminals connected to a mainframe computer.

As the project grew, certain limitations became visible. Users were allowed to provide feedback, but the system initially failed to accurately track posts and even allowed tampering. This security threat was addressed by David Woolley, a young programmer working in CERL, who redesigned the system with what he described as a "linear discussion forum." Woolley's design included:

- Automatic signing of posts with user ID and upload time
- Character count limitation (up to twenty lines)
- Cut-off at sixty-three responses per thread

- Three categories: one for new users, one for developer announcements, and another for general discussions

PLATO's "Notes" feature became the backbone of its community, demonstrating that structured online discussion could foster meaningful interaction and knowledge sharing. Later, PLATO introduced "Talkmatic," designed by Doug Brown in 1973, which enabled real-time chatting among users. This feature partitioned the screen for group members and loaded their keystrokes in real time, making chat feel more realistic despite being limited by 1,200 bits per second bandwidth.

The PLATO system ultimately fell apart when microcomputers launched and proved more cost-effective. However, its legacy lives on in modern learning platforms and discussion forums, having demonstrated core principles that remain relevant today: the importance of user identity, structured conversation, and community moderation.

2.3.2 Email: Overcoming Distance (1971-Present)

The first electronic mail sent from one system to another was accomplished by Raymond Tomlinson, an engineer for Bolt Beranek and Newman (BBN), in 1971 on the ARPANET. Tomlinson modified the SNDMSG program to work with CPYNET, which used the Interface Message Processor (IMP) protocol. He chose the "@" symbol to uniquely identify a local username from a destination computer (Sventek, 1984).

Email's significance for academic communication cannot be overstated. Hauben and Hauben (1997) documented the rise in popularity of email over the years, noting its ability to:

- Enable faster communication than traditional mail
- Overcome location constraints

- Allow groups of people anywhere on the planet to work together by being on the same network
- Greatly improve productivity
- Encourage the use of informal writing styles that bridged formal and casual communication

For universities, email became the primary official communication channel. However, its limitations for collaborative learning particularly the lack of public visibility and searchability have led to the continued search for complementary platforms.

2.3.3 Usenet: The Poor Man's ARPANET (1979-Present)

Usenet began as a project to allow individuals without access to ARPANET to communicate. Initially, it used modems over standard landline telephone lines for data transmission (UUCP protocol), but eventually adopted Network News Transfer Protocol (NNTP) over TCP/IP around 1992.

Usenet is fundamentally a linear discussion forum where subject matters have a many-to-one relationship with newsgroups (modernly called subspaces). A user posts a message tagged to a newsgroup, and others can read and reply. Usenet became the first widely accessible platform for broad community discussion, previously gatekept by ARPANET. What was crudely referred to as "the poor man's ARPANET" became a relevant alternative even for ARPANET users because of its open access (Hauben & Hauben, 1997).

Usenet's hierarchical structure with newsgroups organized by topic and region provided an early model for organizing online discussions that influenced later platforms like Reddit and, indeed, BenTalk's department-and-level-based subspace system.

2.3.4 Web 2.0 Era: Social Media Explosion (2000s-Present)

The early 2000s saw the emergence of social media platforms that would fundamentally change how people communicate online:

Facebook (2004): Originally limited to Harvard students, Facebook expanded to other universities and eventually the general public. Its key features include:

- Personal profiles for self-expression
- Friend connections to show mutual relations
- Groups for communities of interest
- Posts with text, images, and video
- Engagement through comments, likes, and reactions
- Events, marketplace, and notes

While not originally intended for academic use, Facebook has been studied extensively for its educational applications. Tess (2013) found that Facebook can positively impact student engagement, but results are mixed only about half of students in various studies attributed improved academic performance to the platform's use.

YouTube (2005): Google's video-sharing platform became a crucial resource for educational content. While most content is entertainment-focused, high-quality educational channels provide free access to lectures, tutorials, and demonstrations from experts worldwide.

Reddit (2005): A non-linear discussion forum where each sub-community ("subreddit") focuses on specific topics. Reddit prioritizes knowledge sharing over engagement metrics, making it particularly suitable for academic discussions. LaFlamme (2025) describes Reddit as uniquely suitable for digital pedagogy due to its threaded conversation structure and community moderation features.

Twitter/X (2006): A microblogging platform that has become important for scholarly communication, conference backchannel discussions, and rapid dissemination of research findings.

WhatsApp (2009): A mobile messaging platform that has become ubiquitous in developing countries, including Nigeria. Rexwhite and Fasae (2022) document its extensive use in Nigerian universities for creating class groups, sharing announcements, and facilitating group projects. Features include:

- Free synchronous communication (text messaging)
- Audio and video calling
- Voice message transmission
- File and image sharing
- Location sharing
- Group creation

Telegram (2013): Similar to WhatsApp but with additional features like larger group sizes, broadcast channels, and an open-source API that allows for bot development.

2.4 Social Media Platforms in Higher Education

2.4.1 Facebook in Academic Settings

Numerous studies have investigated Facebook's role in higher education. Roblyer et al. (2010) found that faculty members were significantly less likely than students to use Facebook for any purpose and were less favorable toward using it for class-related communication. This faculty-student divide presents a challenge for institutional adoption of Facebook as an academic platform.

Junco (2012) conducted one of the few experimental studies on Facebook use in education, finding that students who were engaged with Facebook in specific, educationally relevant ways showed higher engagement in courses and had higher grades. However, the study also noted that general Facebook use (social browsing) was negatively correlated with academic performance.

The mixed results stem from Facebook's dual nature: it can facilitate academic discussion but is primarily designed for social interaction, leading to frequent distraction and off-topic conversation (Kirschner & Karpinski, 2010).

2.4.2 WhatsApp as a Learning Tool

WhatsApp has become particularly important in the African context due to its low data consumption and widespread mobile phone usage. Rexwhite and Fasae (2022) specifically examined WhatsApp use in Nigerian universities, finding:

Positive Aspects:

- Facilitates quick communication between lecturers and students
- Enables easy sharing of announcements and learning materials
- Supports group collaboration on projects
- Allows for asynchronous communication that accommodates different schedules
- Low barrier to entry (most students already use the app)

Challenges:

- Information overload from excessive messages
- Difficulty finding specific information in long chat histories
- Mixing of academic and social content
- Privacy concerns when sharing personal phone numbers
- Lack of institutional control or moderation

- Inequality issues (students without smartphones are excluded)

Bouhnik and Deshen (2014) identified four main ways teachers use WhatsApp for educational purposes: (1) availability and immediacy (being accessible to students), (2) creating dialogue and encouraging participation, (3) building a social atmosphere that enhances the learning environment, and (4) enabling the learning process itself through content sharing.

2.4.3 YouTube for Educational Content

YouTube has become an invaluable resource for supplementary learning. Jones and Cuthrell (2011) found that students often turn to YouTube tutorials when they struggle with concepts presented in class, appreciating the ability to pause, rewind, and replay explanations at their own pace.

However, the platform's algorithm prioritizes engagement over educational value, and students may encounter inaccurate or misleading information alongside legitimate educational content (Tan & Pearce, 2011). Additionally, YouTube is designed for consumption rather than production of knowledge. Students are passive viewers rather than active participants in knowledge creation.

2.4.4 LinkedIn for Professional Development

While primarily a professional networking platform, LinkedIn has educational components through LinkedIn Learning (formerly Lynda.com). Students use LinkedIn to:

- Build professional profiles showcasing their academic achievements
- Connect with alumni and industry professionals
- Access career resources and job postings
- Demonstrate their expertise through posts and articles

However, LinkedIn's commercial focus and the performative nature of professional networking make it less suitable for authentic academic discourse (Zide et al., 2014).

2.5 Uses of Social Media in Academic Settings

2.5.1 By Faculty and Instructors

Digital Pedagogy and Content Delivery: Instructors use social media platforms to supplement traditional teaching methods. This might include posting lecture recordings on YouTube, sharing articles via Twitter, or facilitating discussions in Facebook groups.

Research Dissemination: Academics use Twitter and LinkedIn to share research findings, engage with peers, and build professional reputations. The use of hashtags enables researchers to participate in broader scholarly conversations.

Student Engagement: Faculty members create closed groups on platforms like Facebook or WhatsApp to maintain communication with students outside of class time, answer questions, and provide clarifications.

Professional Development: Educators connect with colleagues globally, share teaching strategies, and stay current with developments in their fields through professional learning networks on social media.

Van Den Beemt et al. (2020) found that faculty who successfully integrate social media into teaching tend to have clear pedagogical goals, provide structured guidelines for student participation, and actively moderate discussions to maintain academic focus. However, many instructors struggle with establishing appropriate boundaries and managing the time commitment required to maintain an active social media presence for educational purposes (Macht et al., 2020).

2.5.2 By Students

Information Seeking and Knowledge Sharing: Students use social media to find answers to questions, seek clarification on concepts, and share resources with peers. The ability to access multiple perspectives and explanations helps accommodate different learning styles.

Collaborative Learning: Group projects are often coordinated through WhatsApp or Facebook Messenger, where students can easily share files, discuss ideas, and coordinate meeting times.

Peer Support and Mentoring: Informal mentoring relationships often develop through social media, with senior students providing guidance and advice to juniors about courses, career paths, and university life in general.

Academic Networking: Students build connections with peers in their field of study, both within their institution and globally. These networks can lead to research collaborations, study abroad opportunities, and career connections.

Self-Promotion and Portfolio Building: Students showcase their work, achievements, and developing expertise through social media posts, potentially attracting attention from potential employers or graduate school admission committees.

Zachos et al. (2018) found that students who actively engaged with academic content on social media demonstrated higher levels of self-efficacy and were more likely to persist in challenging courses. The visibility of their learning process and the feedback received from peers provided motivation and validation.

2.6 Challenges of Using Social Media for Academic Purposes

Despite the potential benefits, significant challenges limit the effectiveness of general social media platforms for academic use:

2.6.1 Information Deluge and Content Fragmentation

Macht et al. (2020) describe the overwhelming volume of content on social media as a major barrier to effective use. Important academic posts are quickly buried under social content, memes, and spam. Students report spending excessive time scrolling through irrelevant content to find the information they need.

The fragmentation across multiple platforms exacerbates this problem. A student might need to check WhatsApp for class announcements, Facebook for study group discussions, email for official university communications, and Twitter for updates from their lecturer all while trying to remember where they saw that important piece of information last week.

2.6.2 Lack of Institutional Oversight and Moderation

Most academic uses of social media occur in spaces created by students or individual faculty members, without institutional involvement or oversight. This creates several problems:

- **Misinformation**: Without moderation, false or misleading information can spread unchecked (Van Den Beemt et al., 2019).
- **Impersonation**: There's no verification of identity, allowing bad actors to impersonate students or faculty.
- **Lack of Accountability**: No official record exists of what information was shared or who shared it.
- **No Continuity**: When a faculty member leaves or a student graduates, their social media groups may become inactive, and valuable content is lost.

2.6.3 Erosion of Professional Boundaries

The informal nature of social media, combined with its integration into personal life, creates uncomfortable situations for both students and faculty. Macht et al. (2020) document how

sharing personal contact information (phone numbers for WhatsApp, personal Facebook profiles) blurs the line between professional and personal relationships.

Faculty members report feeling pressure to be constantly available to students and experiencing difficulty "switching off" from work. Students, meanwhile, may feel uncomfortable reaching out to instructors on personal platforms or may receive unwanted contact from peers in their personal social media spaces.

Van Den Beemt et al. (2019) emphasize that teachers struggle to maintain appropriate authority and professionalism in social media spaces designed for casual interaction. The same platform where students share party photos is not conducive to serious academic discourse.

2.6.4 Privacy and Security Concerns

Social media platforms collect extensive data about users and use this data for advertising purposes. Students and faculty have legitimate concerns about:

- **Data Mining**: How is their academic communication being analyzed and used?
- **Third-Party Access**: Who else can see their posts and personal information?
- **Long-Term Consequences**: Will casual social media posts from their student days affect future employment prospects?
- **Institutional Data**: Universities have no control over their own institutional knowledge when it's hosted on external platforms.

These concerns are particularly acute in developing countries like Nigeria, where data protection regulations may be less stringent and enforced (Adewale & Aremu, 2020).

2.6.5 Algorithm Misalignment with Educational Goals

Social media algorithms are designed to maximize engagement measured by likes, shares, comments, and time spent on the platform. This creates a system where:

- Controversial or emotional content is prioritized over substantive academic discussion
- Educational posts with fewer interactions sink in visibility
- Students are constantly distracted by non-academic content engineered to capture attention
- The most popular content is not necessarily the most accurate or educationally valuable

As Facebook CEO Mark Zuckerberg acknowledged, only a small percentage of content users see is from accounts they actually follow; most is selected by the algorithm to maximize engagement (Securities and Exchange Commission, 2022).

2.6.6 Digital Divide and Accessibility Issues

While social media platforms are generally free, they still present accessibility challenges:

- **Data Costs**: In Nigeria and other developing countries, mobile data is expensive relative to income. Video-heavy platforms like Facebook drain data quickly.
- **Device Requirements**: Some platforms perform poorly on older or budget smartphones.
- **Internet Connectivity**: Unreliable internet infrastructure means students may not be able to access platforms consistently.
- **Digital Literacy**: Not all students are equally comfortable navigating complex social media platforms or distinguishing credible information from misinformation.

These issues can exacerbate existing inequalities, with disadvantaged students unable to participate fully in online academic communities (Adewale & Aremu, 2020).

2.6.7 Resistance from Faculty

Despite evidence of potential benefits, many faculty members remain resistant to using social media for academic purposes. Tess (2013) and Macht et al. (2020) identify several reasons:

- **Time Constraints**: Managing social media presence requires significant time investment

- **Lack of Training**: Faculty may not feel confident using platforms effectively
- **Philosophical Opposition**: Some believe social media is fundamentally incompatible with academic rigor
- **Institutional Pressure**: Using unofficial platforms may conflict with university policies
- **Negative Experiences**: Faculty who have tried and struggled with social media may become discouraged

Van Den Beemt et al. (2019) document multiple instances where teachers attempted to integrate social media but abandoned the effort due to students' unprofessional engagement, inability to control discussions, and failure of students to conduct themselves formally. This tension between the need for structured academic discourse and the informal nature of social media remains unresolved on general platforms.

2.7 Review of Related Systems

2.7.1 Learning Management Systems (LMS)

Traditional Learning Management Systems like Moodle, Blackboard, and Canvas provide structured environments for online education:

Strengths:

- Institutional control and oversight
- Integration with student records and grades
- Structured organization of course materials
- Clear delineation between different courses
- Privacy and security by design
- Accessibility features for students with disabilities

Weaknesses:

- Often perceived as bureaucratic and inflexible by students

- Limited social interaction features
- Typically course-specific rather than community-wide
- May require institutional licenses and IT support
- Often have outdated user interfaces
- Not designed for informal peer-to-peer interaction

LMS platforms excel at delivering structured course content but fail to replicate the spontaneous, community-driven knowledge sharing that happens naturally on social media platforms.

2.7.2 Reddit and Subreddit Communities

Reddit provides a model of community-driven discussion that has proven effective for learning:

Strengths:

- Topic-specific subreddits create focused communities
- Voting system surfaces high-quality content
- Threaded conversations maintain context
- Searchable archive of past discussions
- Anonymity allows for honest questions without fear of judgment
- Large user base means questions are often answered quickly

Weaknesses:

- No institutional oversight or verification of information accuracy
- Anonymous nature can lead to trolling or harassment
- Global focus means content may not be relevant to specific institutional contexts
- Overwhelming volume of content in large subreddits
- No integration with university systems or official communications

LaFlamme (2025) argues that Reddit's structure makes it uniquely suitable for academic use, noting that students often turn to subject-specific subreddits (e.g., r/learnprogramming, r/AskHistorians) for explanations and discussions that complement formal education.

2.7.3 Discord for Academic Communities

Discord, originally designed for gaming communities, has been increasingly adopted by academic groups:

Strengths:

- Real-time chat and voice communication
- Channel-based organization keeps different topics separate
- Supports both synchronous and asynchronous communication
- Free to use with robust free tier
- Integration with other services through bots
- Mobile and desktop applications

Weaknesses:

- Primarily synchronous older messages are hard to find
- No built-in voting or quality signaling mechanisms
- Can feel chaotic and overwhelming to new users
- Not designed specifically for knowledge preservation
- Lacks academic-specific features like citation tools or formal Q&A structure

2.7.4 Piazza for Course Q&A

Piazza is a platform specifically designed for classroom Q&A:

Strengths:

- Instructors can endorse correct answers

- Anonymous posting options
- Email notifications keep users engaged
- Syntax highlighting for code
- Integrates with LMS platforms
- Specifically designed for academic use

Weaknesses:

- Typically course-specific, not community-wide
- Requires instructor setup and management
- Limited social features
- Not designed for informal community building
- May require institutional licensing

2.7.5 Slack for Academic Workspaces

Slack provides team communication tools that some institutions have adapted for academic use:

Strengths:

- Channel-based organization
- Searchable message history
- File sharing and integration with other tools
- Professional, business-like interface
- Strong mobile applications

Weaknesses:

- Free tier has limited message history
- Can become expensive for large institutions
- Designed for workplace communication, not learning

- May feel too formal for casual student interaction
- Synchronous nature means messages can be missed

2.8 Gap Analysis and Justification for BenTalk

After reviewing existing literature and systems, several gaps become apparent:

2.8.1 Context-Specific Solutions for Nigerian Universities

Most platforms reviewed are designed for Western contexts with reliable high-speed internet, expensive devices, and specific institutional structures. Nigerian universities face unique challenges:

- **Infrastructure Limitations**: Inconsistent internet connectivity, limited bandwidth, frequent power outages
- **Device Constraints**: Many students access the internet primarily through budget smartphones
- **Cultural Context**: Need for platforms that respect local academic hierarchies and communication norms
- **Economic Factors**: Preference for low-data consumption and free platforms

BenTalk addresses these by providing a lightweight, mobile-first design optimized for low-bandwidth environments and respecting the departmental/level structure common in Nigerian universities.

2.8.2 Balance Between Formal and Informal Communication

Existing systems tend to fall into two extremes:

- **Too Formal**: LMS platforms feel bureaucratic and discourage spontaneous discussion
- **Too Informal**: Social media platforms lack structure and professional boundaries

BenTalk aims to strike a balance by providing structured subspaces (departments and levels) while allowing informal peer-to-peer interaction within those boundaries. The platform maintains academic focus without the rigidity of traditional LMS systems.

2.8.3 Institutional Oversight Without Centralized Control

While students need freedom to create and manage their own discussions, universities need some oversight to ensure information quality and community standards. Most current solutions offer either:

- **Full institutional control** (LMS platforms) that stifles student ownership
- **No institutional presence** (social media groups) that leads to misinformation and lack of continuity

BenTalk's design allows for optional moderator roles for faculty while keeping primary control with the student community.

2.8.4 Knowledge Preservation and Searchability

Social media platforms like WhatsApp and Facebook treat messages as ephemeral valuable discussions disappear into chat histories. Reddit and forums preserve content but lack specific organizational structure for academic hierarchies.

BenTalk provides permanent, searchable archives organized by department and level, ensuring that today's answers remain accessible to future students with similar questions.

2.8.5 Community-Wide Rather Than Course-Specific

LMS platforms and tools like Piazza are typically organized around individual courses, missing the broader student community that exists across courses and semesters. BenTalk creates a persistent community space that transcends individual course boundaries while respecting departmental and level divisions.

2.9 Summary of Literature Review

The literature review reveals that social media has become integral to higher education despite not being designed for academic purposes. Platforms like Facebook, WhatsApp, and YouTube provide valuable tools for communication, collaboration, and content sharing. However, significant challenges persist:

1. **Fragmentation**: Academic discourse is scattered across multiple platforms
2. **Information Overload**: Important content is buried under social noise
3. **Lack of Structure**: General platforms don't align with academic hierarchies
4. **Privacy Concerns**: Personal data and academic discussions exist on commercial platforms
5. **Boundary Erosion**: Mixing of personal and professional spaces
6. **Algorithm Misalignment**: Engagement-focused rather than education-focused
7. **No Institutional Oversight**: Universities have no control over student discussions

Historical precedents from PLATO and Usenet demonstrate that structured, purpose-built platforms can effectively support academic communities. Modern platforms like Reddit show that community-driven, threaded discussions work well for knowledge sharing. However, none of the existing solutions adequately addresses the specific needs of Nigerian universities.

BenTalk is positioned to fill this gap by providing a dedicated, structured platform that preserves the best features of social media (ease of use, community-driven content, mobile accessibility) while addressing the limitations through institutional oversight, academic structure, and knowledge preservation. The next chapter will analyze existing systems in the University of Benin context and detail the design of the proposed solution.

CHAPTER THREE

SYSTEM ANALYSIS AND DESIGN

3.1 Analysis of the Existing System

Currently, students at the University of Benin, particularly within the Faculty of Computing, rely on a fragmented ecosystem of communication tools for academic purposes:

3.1.1 Current Communication Channels

WhatsApp Groups: The most widely used platform for academic communication. Students create numerous groups:

- Course-specific groups (e.g., "CSC 301 Group")
- Level-based groups (e.g., "Computer Science 300L 2024/2025")
- Department-wide groups
- Project team groups
- Study groups for specific topics

Characteristics:

- Immediate, synchronous communication
- Easy file and image sharing
- Audio and video conference capabilities
- Low data consumption
- Near-universal student adoption

Facebook Groups: Some departments maintain Facebook groups for broader discussions and announcements.

Email: Used primarily for formal communications from lecturers and university administration, but often overlooked by students who don't check regularly.

Physical Notice Boards: Still used for official announcements, but require students to physically visit campus.

Word of Mouth: Much information still spreads through informal face-to-face conversations.

3.1.2 Information Flow in Current System

The typical pattern for academic information dissemination follows this fragmented path:

1. **Official Announcement**: Lecturer posts information on physical notice board or sends email
2. **Student Intermediaries**: Active students who check these sources relay information to WhatsApp groups
3. **Multi-Platform Propagation**: Information spreads across multiple WhatsApp groups with varying accuracy
4. **Questions and Confusion**: Students ask questions in multiple groups, receiving different answers
5. **Information Loss**: After a few days, the information becomes difficult to find in chat histories
6. **Repeated Questions**: New students or those who missed the initial announcement ask the same questions again

This creates an inefficient system where:

- The same information exists in multiple places with potential inconsistencies
- Important details are lost or distorted through repeated sharing
- Student time is wasted searching through chat histories or asking redundant questions
- Lecturers answer the same questions multiple times across different channels
- No authoritative source exists for common questions

3.2 Constraints of the Existing System

3.2.1 Technical Constraints

Lack of Searchability: WhatsApp's search function is limited to keyword matching within long, chronological chat histories. Finding specific information from weeks or months ago is extremely difficult.

No Content Organization: Messages appear in strict chronological order regardless of topic. A conversation about something as critical as continuous assessment might be interrupted by questions about the exam, various social activities, and memes, making it difficult to follow any single discussion thread.

Message Ephemerality: Important information gets quickly buried under new messages. Students who miss a day of WhatsApp activity may never see critical announcements.

File Management Issues: Files shared in WhatsApp are not organized or labeled systematically. Students struggle to find specific documents later and may need to download them multiple times, consuming data.

Group Size Limits: WhatsApp allows a large number, but still exercises limits in the allowed total group participants, potentially forcing larger cohorts to split into multiple groups, fragmenting discussions further.

3.2.2 Social and Organizational Constraints

Mixing of Content Types: Academic discussions, social conversations, spam, and off-topic content coexist in the same space, making it difficult to maintain focus.

Lack of Moderation: No formal moderation means spam, misinformation, and inappropriate content can spread unchecked. While group admins have some control, they're typically fellow students without clear authority or guidelines.

Unclear Boundaries: Students may feel pressured to respond to academic questions during personal time. The 24/7 nature of WhatsApp creates expectations of constant availability.

Exclusion of Faculty: Most student WhatsApp groups exclude faculty members, meaning lecturers cannot provide authoritative answers or correct misinformation directly.

Privacy Issues: Sharing personal phone numbers with potentially hundreds of group members raises privacy and security concerns.

Language Barriers: Discussions often mix English, Pidgin English, and local languages, which may exclude some students or create confusion in technical discussions.

3.2.3 Institutional Constraints

No Official Oversight: The university has no visibility into what information is being shared in student groups and cannot correct misinformation or provide official clarifications.

Lack of Continuity: When students graduate, their knowledge and contributions leave with them. Each new cohort must build information resources from scratch.

No Integration: These informal communication channels don't integrate with official university systems, creating disconnects between formal and informal information sources.

Inability to Analyze: The university cannot analyze common student questions or concerns to improve teaching or address systemic issues.

3.3 Justification of the Proposed System

Given the constraints identified above, BenTalk is justified by the following needs:

3.3.1 Centralization of Academic Discourse

A single platform where students can reliably find and contribute academic information eliminates the need to check multiple platforms and reduces information fragmentation. This centralization should:

- Reduce time spent searching for information
- Ensure consistency of information
- Create a single authoritative source for department-specific questions
- Facilitate knowledge transfer between cohorts

3.3.2 Structured Information Architecture

Organizing discussions by department and level creates natural boundaries that align with the university's academic structure. This provides:

- Relevant content for each student's specific context
- Clear navigation paths to find pertinent information
- Reduced noise from irrelevant discussions
- Alignment with existing mental models of university organization

3.3.3 Knowledge Preservation

Unlike ephemeral chat messages, a permanent archive ensures that valuable discussions remain accessible. This provides:

- Reduced redundancy (fewer repeated questions)
- Learning resource for future students
- Institutional memory that persists across cohorts
- Evidence of common challenges that can inform curriculum improvements

3.3.4 Quality Signaling Mechanisms

Upvote/downvote functionality allows the community to surface high-quality contributions while filtering out less useful content. This creates:

- Self-moderating content quality
- Recognition for helpful contributors

- Reduced cognitive load (students can focus on highly-rated content)
- Incentives for thoughtful, accurate contributions

3.3.5 Optional Institutional Oversight

Providing roles for faculty moderators (without requiring their constant presence) enables:

- Authoritative answers when needed
- Correction of misinformation
- Official announcements in appropriate contexts
- Bridge between formal and informal communication
- Maintained student ownership of discussions

3.3.6 Privacy and Security

A university-specific platform provides:

- Control over who can access discussions
- No sharing of personal contact information required
- University authentication ensuring users are who they claim to be
- Institutional data sovereignty
- Clear terms of service and acceptable use policies

3.3.7 Mobile-First, Low-Bandwidth Design

Recognizing the constraints of Nigerian infrastructure, BenTalk is designed to:

- Function on budget smartphones
- Minimize data consumption
- Provide offline reading capabilities where possible
- Maintain performance on slow connections
- Prioritize text-based communication over media-heavy content

3.4 System Requirements

3.4.1 Functional Requirements

The system must provide the following functionalities:

User Management:

- FR1: Users shall be able to register accounts with username, email, matric ID, department, and level
- FR2: Users shall authenticate using username/email and password
- FR3: User profiles shall display academic information (department, level, faculty)
- FR4: Users shall be able to update their profile information
- FR5: System shall maintain session state across page refreshes and app restarts

Navigation and Subspaces:

- FR6: Users shall be able to view all departments in the Faculty of Computing
- FR7: Users shall be able to select a department to view its levels (100L, 200L, 300L, 400L)
- FR8: Users shall be able to select a level to view posts specific to that department-level combination
- FR9: System shall maintain breadcrumb navigation showing current location hierarchy

Post Creation and Management:

- FR10: Users shall be able to create posts with title and content
- FR11: Posts shall be automatically tagged with author information and timestamp
- FR12: Users shall be able to edit their own posts within a time limit
- FR13: Users shall be able to delete their own posts
- FR14: System shall maintain post history for moderation purposes

Comments and Discussion:

- FR15: Users shall be able to comment on posts
- FR16: Users shall be able to reply to existing comments (nested threading)
- FR17: Comments shall display author and timestamp
- FR18: Users shall be able to edit their own comments within a time limit
- FR19: Users shall be able to delete their own comments

Voting System:

- FR20: Users shall be able to upvote posts they find helpful
- FR21: Users shall be able to downvote posts they find unhelpful
- FR22: Users shall be able to change or remove their votes
- FR23: System shall display aggregate vote scores for each post
- FR24: Posts shall be orderable by vote score

Real-Time Updates:

- FR25: Users shall receive real-time notifications of new posts in viewed subspaces
- FR26: Comment sections shall update in real-time when new comments are added
- FR27: Vote counts shall update in real-time
- FR28: System shall indicate when other users are typing comments

Moderation (Future Enhancement):

- FR29: Faculty moderators shall be able to pin important posts
- FR30: Moderators shall be able to remove inappropriate content
- FR31: Moderators shall be able to mark posts as "answered" or "official"
- FR32: System shall maintain moderation logs

3.4.2 Non-Functional Requirements

Performance:

- NFR1: Web pages shall load within 3 seconds on a 3G connection
- NFR2: API responses shall complete within 500ms under normal load
- NFR3: System shall support 500 concurrent users without degradation
- NFR4: Database queries shall be optimized with appropriate indexing

Usability:

- NFR5: First-time users shall be able to create a post within 5 minutes without training
- NFR6: Navigation shall require no more than 3 clicks to reach any content
- NFR7: Interface shall be responsive across devices (desktop, tablet, mobile)
- NFR8: Text shall be readable without zooming on mobile devices (responsive)
- NFR9: Color scheme shall provide sufficient contrast for readability

Reliability:

- NFR10: System shall have 99% uptime during operating hours
- NFR11: Data shall be backed up daily with point-in-time recovery capability
- NFR12: System shall gracefully handle network interruptions
- NFR13: Failed operations shall provide clear error messages and recovery options

Security:

- NFR14: Passwords shall be hashed using bcrypt with appropriate salt rounds
- NFR15: Authentication tokens shall expire after 24 hours
- NFR16: All API endpoints shall require authentication except public read-only access
- NFR17: Input validation shall prevent SQL injection and XSS attacks
- NFR18: HTTPS shall be enforced in production environment

Maintainability:

- NFR19: Code shall follow PEP 8 style guidelines (Python) and standard conventions (JavaScript, Kotlin)
- NFR20: API shall be fully documented with OpenAPI/Swagger
- NFR21: Database schema shall be version-controlled with migrations
- NFR22: System shall provide logging for debugging and monitoring

Portability:

- NFR23: Backend shall run on Linux, Windows, and macOS
- NFR24: Web interface shall function on Chrome, Firefox, Safari, and Edge
- NFR25: Android app shall support devices running Android 10 and above
- NFR26: Database shall use standard SQL compatible with PostgreSQL

Scalability:

- NFR27: Architecture shall support horizontal scaling of application servers
- NFR28: Database shall support read replicas for query distribution
- NFR29: Static assets shall be servable from CDN
- NFR30: System design shall accommodate future feature additions without major refactoring

3.5 Proposed System Architecture

BenTalk follows a three-tier architecture pattern with clear separation of concerns:

3.5.1 Presentation Tier (Client)

Web Application (React):

- Single-page application (SPA) architecture
- Component-based UI with reusable elements

- Client-side routing for seamless navigation
- State management for application data
- Local storage (Dexie.js/IndexedDB) for offline capabilities
- Responsive design adapting to screen sizes

Mobile Application (Kotlin/Android):

- Native Android application
- Jetpack Compose for modern UI development
- MVVM (Model-View-ViewModel) architecture pattern
- Retrofit for network communication
- Room database for local caching
- Material Design 3 guidelines

3.5.2 Application Tier (Business Logic)

FastAPI Backend:

- RESTful API architecture
- Asynchronous request handling
- Automatic API documentation (OpenAPI/Swagger)
- Input validation using Pydantic models
- JWT-based authentication
- WebSocket support for real-time updates
- CORS middleware for cross-origin requests

Key Modules:

- **Authentication Module**: User registration, login, token management
- **User Module**: Profile management, user queries

- **Post Module**: CRUD operations for posts, search functionality
- **Comment Module**: CRUD operations for comments, nested reply handling
- **Vote Module**: Vote casting, score calculation, vote state management
- **Subspace Module**: Department and level organization
- **WebSocket Module**: Real-time broadcast of updates

3.5.3 Data Tier (Persistence)

PostgreSQL Database:

- Relational database management system
- ACID-compliant transactions
- Complex query support with indexing
- Full-text search capabilities
- JSON data type support for flexible fields
- Connection pooling for performance

Database Schema Components:

- Users table: account information, authentication credentials
- Posts table: post content, metadata, foreign keys to users and subspaces
- Comments table: comment content, threading structure, references
- Votes table: user votes on posts, composite keys
- Subspaces table: department and level hierarchy
- Followers table: user-subspace relationships (future feature)

3.5.4 Architecture Diagram

Tier	Components	Description
Presentation Tier	React Web App • Components • State Management • Dexie.js Storage • React Router Kotlin Android App • Jetpack Compose UI • ViewModels • Room Database • Navigation Component	Handles UI/UX, manages state, and interacts with backend via HTTP/WebSocket.
Application Tier	FastAPI Backend • Auth Routes • Post Routes • Comment Routes • User Routes • Vote Routes • Subspace Routes • WebSocket Connection Manager • SQLAlchemy ORM Layer	Processes requests, business logic, manages data transactions, and handles WebSocket connections.
Data Tier	PostgreSQL Database • Users • Posts • Comments • Votes • Subspaces • subspace_followers (M:M)	Stores and retrieves structured data persistently with relational integrity.

3.6 System Design

3.6.1 Use Case Diagrams

Primary Actors:

- Student (unregistered)
- Student (registered)
- Lecturer
- Administrator

Key Use Cases:

Table 3.6.1 – BenTalk Use Cases

Actor	Use Cases
Student (Unregistered)	Browse Departments, View Posts (Read Only), Search Posts
Student (Registered)	Sign Up, Login, Browse Departments & Levels, Create Post, Edit Own Post, Delete Own Post, Add Comment, Reply to Comment, Upvote/Downvote Post, Search Posts, View Profile, Update Profile
Lecturer	All Student Capabilities, Pin Important Posts, Mark Post as Official, Moderate Content
Administrator	Manage Users, Manage Subspaces, View System Analytics, Configure System Settings

Future enhancement features are planned but not mandatory for current implementation.

3.6.2 Data Flow Diagrams

Table 3.6.2: Data Flow Diagrams

This table summarizes the main processes and data flow levels represented in the system's DFDs.

DFD Level	Description
Level 0 (Context Diagram)	Shows external entities such as Students and Lecturers interacting with the BenTalk System through posts, searches, notifications, and moderation. The database is the central data store.
Level 1 (Authentication)	Depicts the process of user login and authentication, resulting in the generation of a user token.
Level 1 (Subspace Navigation)	Shows user requests for department or level subspaces, with corresponding database queries and results.
Level 1 (Post Management)	Covers creation, editing, and deletion of posts stored in the database with confirmation feedback.
Level 1 (Comment Management)	Details adding, replying to, and managing comments stored in the database with confirmation feedback.
Level 1 (Voting System)	Illustrates users voting on posts, updating scores in the database, and reflecting new scores in the interface.
Level 1 (Search Engine)	Explains user queries for posts and retrieval of matching results from the database.
Level 1 (Real-time Broadcast)	Shows WebSocket Manager broadcasting updates to connected clients for real-time communication.

3.6.3 Entity-Relationship Diagram

This table summarizes the main entities, attributes, and relationships within the BenTalk system.

Entity	Attributes	Relationships
Users	PK: id (INT) username, full_name, matric_id, email, password_hash, faculty, department, session, level, bio, profile_picture, joined_at	User (1) ──< Posts (N) User (1) ──< Comments (N) User (1) ──< Votes (N) User (M) ──< subspace_followers >──< Subspaces (N)
Posts	PK: id (INT) FK: user_id (INT), FK: subspace_id (INT) title, content, created_at	Post (1) ──< Comments (N) Post (1) ──< Votes (N) Subspace (1) ──< Posts (N)
Comments	PK: id (INT) FK: post_id (INT), FK: user_id (INT), FK: parent_id (INT, NULL) content, created_at	Comment (1) ──< Comments (N) (self-referential)
Subspaces	PK: id (INT) FK: parent_id (INT, NULL) name, type, description, created_at	Subspace (1) ──< Posts (N)
Votes	PK: id (INT) FK: post_id (INT), FK: user_id	

	(INT) value (INT, -1/+1) UNIQUE(post_id, user_id)	
subspace_followers (M:M)	FK: subspace_id (INT), FK: user_id (INT) PRIMARY KEY (subspace_id, user_id)	Many-to-Many relationship between Users and Subspaces

Relationships:

- User (1) ──< Posts (N): One user can create many posts
- User (1) ──< Comments (N): One user can write many comments
- Post (1) ──< Comments (N): One post can have many comments
- Comment (1) ──< Comments (N): One comment can have many reply comments (self-referential)
- Subspace (1) ──< Posts (N): One subspace contains many posts
- Post (1) ──< Votes (N): One post can have many votes
- User (1) ──< Votes (N): One user can cast many votes
- User (M) ──< subspace_followers >──< Subspaces (N): Many-to-Many relationship (users can follow multiple subspaces)

3.7 Database Design

3.7.1 Table Specifications

Users Table:

```
CREATE TABLE users (
    id SERIAL PRIMARY KEY,
```

```
    username VARCHAR(50) UNIQUE NOT NULL,
    full_name VARCHAR(100) NOT NULL,
    matric_id VARCHAR(20) UNIQUE NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    password VARCHAR(255) NOT NULL,
    faculty VARCHAR(100) NOT NULL,
    department VARCHAR(100) NOT NULL,
    session VARCHAR(22) NOT NULL,
    level VARCHAR(10) NOT NULL,
    bio TEXT,
    profile_picture_url VARCHAR(255),
    joined_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    INDEX idx_username (username),
    INDEX idx_email (email),
    INDEX idx_matric_id (matric_id)
);
```

Subspaces Table:

```
CREATE TABLE subspace (
    id SERIAL PRIMARY KEY,
    name VARCHAR(100) UNIQUE NOT NULL,
    type VARCHAR(50) NOT NULL,
    parent_id INTEGER REFERENCES subspace(id),
    description TEXT,
```

```
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    INDEX idx_name (name),
    INDEX idx_type (type)
);
```

Posts Table:

```
CREATE TABLE posts (
    id SERIAL PRIMARY KEY,
    user_id INTEGER NOT NULL REFERENCES users(id) ON DELETE CASCADE,
    subspace_id INTEGER REFERENCES subspace(id),
    title VARCHAR(255) NOT NULL,
    content TEXT NOT NULL,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    INDEX idx_user_id (user_id),
    INDEX idx_subspace_id (subspace_id),
    INDEX idx_created_at (created_at),
    INDEX idx_title_content (title, content) -- For search
);
```

Comments Table:

```
CREATE TABLE comments (
    id SERIAL PRIMARY KEY,
    post_id INTEGER NOT NULL REFERENCES posts(id) ON DELETE CASCADE,
    user_id INTEGER NOT NULL REFERENCES users(id) ON DELETE CASCADE,
    parent_id INTEGER REFERENCES comments(id) ON DELETE CASCADE,
```

```
    content TEXT NOT NULL,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    INDEX idx_post_id (post_id),
    INDEX idx_user_id (user_id),
    INDEX idx_parent_id (parent_id)
);
```

Votes Table:

```
CREATE TABLE votes (
    id SERIAL PRIMARY KEY,
    post_id INTEGER NOT NULL REFERENCES posts(id) ON DELETE CASCADE,
    user_id INTEGER NOT NULL REFERENCES users(id) ON DELETE CASCADE,
    value INTEGER NOT NULL CHECK (value IN (-1, 1)),
    UNIQUE (post_id, user_id),
    INDEX idx_post_id (post_id)
);
```

Subspace Followers Table (Many-to-Many):

```
CREATE TABLE subspace_followers (
    subspace_id INTEGER NOT NULL REFERENCES subspace(id) ON DELETE CASCADE,
    user_id INTEGER NOT NULL REFERENCES users(id) ON DELETE CASCADE,
    PRIMARY KEY (subspace_id, user_id)
);
```

3.7.2 Normalization

The database design adheres to Third Normal Form (3NF):

First Normal Form (1NF):

- All tables have primary keys
- All columns contain atomic values (no arrays or nested structures)
- Each column contains values of a single type

Second Normal Form (2NF):

- All tables are in 1NF
- All non-key attributes are fully functionally dependent on the primary key
- No partial dependencies exist

Third Normal Form (3NF):

- All tables are in 2NF
- No transitive dependencies exist
- All non-key attributes depend only on the primary key

3.7.3 Indexing Strategy

Strategic indexes are placed on:

- **Foreign keys**: For efficient join operations
- **Frequently queried columns**: username, email, matric_id for user lookups
- **Search columns**: title and content for text search operations
- **Timestamp columns**: created_at for chronological ordering
- **Composite unique constraints**: (post_id, user_id) in votes table to prevent duplicate votes

3.7.4 Data Integrity Constraints

Primary Key Constraints:

- Ensure each record is uniquely identifiable
- Auto-incrementing serial IDs for all tables

Foreign Key Constraints:

- Maintain referential integrity between related tables
- CASCADE deletes ensure orphaned records are removed

Unique Constraints:

- username, email, matric_id must be unique in users table
- (post_id, user_id) combination must be unique in votes table

Check Constraints:

- vote value must be either -1 or 1
- Email format validation at application layer

NOT NULL Constraints:

- Critical fields like username, password, post title cannot be null

CHAPTER FOUR

SYSTEM IMPLEMENTATION

4.1 Introduction

This chapter details the implementation of BenTalk, covering the development environment, technologies used, coding practices, and challenges encountered. The system was implemented in phases: backend development, web frontend development, and mobile application development.

The implementation follows modern software engineering practices including:

- Version control using Git
- Modular code organization
- API-first design approach
- Test-driven development principles
- Documentation-as-you-go methodology

4.2 Hardware Requirements

4.2.1 Development Environment

Minimum Requirements:

- Processor: Intel Core i5 or AMD Ryzen 5 (or equivalent)
- RAM: 8GB minimum, 16GB recommended
- Storage: 256GB SSD with at least 50GB free space
- Display: 1920x1080 resolution minimum
- Network: Stable internet connection for package downloads and testing

Development Machine Used:

- Processor: Intel(R) Core(™) i7-1065G7 CPU @ 1.50GHz
- RAM: 16 Gigabytes

- Storage: 1 Terabyte
- Operating System: Windows 11

4.2.2 Production Deployment Requirements

Minimum Server Specifications:

- CPU: 2 cores
- RAM: 4GB
- Storage: 50GB SSD
- Network: 100 Mbps connection
- Operating System: Ubuntu 20.04 LTS or later

Recommended for 500+ Concurrent Users:

- CPU: 4+ cores
- RAM: 8GB+
- Storage: 100GB+ SSD with regular backups
- Network: 1 Gbps connection
- Load balancer for horizontal scaling

4.2.3 Client Device Requirements

Web Application:

- Any device with a modern web browser
- Minimum screen resolution: 320px width
- JavaScript enabled

Android Mobile Application:

- Android 10 (API level 29) or higher
- Minimum 2GB RAM

- 100MB free storage
- Internet connectivity (WiFi or mobile data)

4.3 Software Requirements

4.3.1 Backend Development

Core Framework:

- **Python 3.9+**: Programming language for backend
- **FastAPI 0.116.1**: Web framework for building APIs
- **Uvicorn 0.35.0**: ASGI server for running FastAPI
- **SQLAlchemy 2.0.42**: ORM for database operations
- **PostgreSQL 13+**: Relational database management system

Additional Python Packages:

- **pydantic 2.11.7**: Data validation using Python type annotations
- **python-jose**: JWT token creation and verification
- **passlib**: Password hashing utilities
- **python-multipart**: Form data parsing
- **psycopg2-binary**: PostgreSQL adapter for Python

Development Tools:

- **Git**: Version control
- **Postman**: API testing
- **pgAdmin**: PostgreSQL database management
- **VS Code**: Code editor/IDE

4.3.2 Web Frontend Development

Core Technologies:

- **React 18.2+**: JavaScript library for building user interfaces

- **React Router 6.x**: Client-side routing
- **Dexie.js 3.x**: IndexedDB wrapper for browser storage

Build Tools:

- **Vite**: Build tool and development server
- **npm/yarn**: Package manager

Development Environment:

- **Node.js 16+**: JavaScript runtime
- **Chrome DevTools**: Debugging and performance profiling
- **VS Code**: Code editor with React extensions

4.3.3 Mobile Application Development

Core Technologies:

- **Kotlin 2.0.21**: Programming language for Android
- **Jetpack Compose**: Modern UI toolkit
- **Android Studio**: Integrated development environment

Android Libraries:

- **Retrofit 2.11.0**: HTTP client for API communication
- **Gson 2.11.0**: JSON serialization/deserialization
- **OkHttp 4.12.0**: HTTP client for networking
- **Navigation Component 2.8.0**: Navigation framework
- **Coroutines 1.9.0**: Asynchronous programming

Build Configuration:

- **Gradle**: Build automation tool
- **Android SDK 36**: Target API level

- **Minimum SDK 29**: Android 10+

4.3.4 Database Management

- **PostgreSQL 13+**: Production database
- **SQLite 3**: Development and testing database
- **Alembic**: Database migration tool
- **Database Backup Tools**: pg_dump, pg_restore

4.4 System Development

4.4.1 Backend Implementation (Python/FastAPI)

Project Structure:

4.4.1 Backend Implementation (Python/FastAPI)

Path	Description
bentalk-backend/	Root directory for the FastAPI backend project.
├── main.py	Application entry point initializes FastAPI app and routes.
├── database.py	Database connection and SQLAlchemy session management.
├── model.py	Defines SQLAlchemy ORM models for Users, Posts, Comments, etc.
├── schema.py	Defines Pydantic models for request/response validation.
├── auth.py	Authentication logic password hashing, JWT handling, verification.
├── auth_routes.py	Routes for user registration, login, and authentication endpoints.
├── requirements.txt	Lists Python dependencies for the backend environment.
└── alembic/	Contains database migration scripts and configurations.
├── versions/	Individual migration files for schema evolution.
└── env.py	Alembic environment configuration for migrations.

Core Implementation Details:

1. Database Configuration (database.py):

```
from sqlalchemy import create_engine
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import sessionmaker
DATABASE_URL = "postgresql://user:password@localhost/bentalk"
engine = create_engine(DATABASE_URL)
SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)
Base = declarative_base()
def get_db():
    db = SessionLocal()
    try:
        yield db
    finally:
        db.close()
```

Key Features:

- Connection pooling for efficient database access
- Dependency injection for database sessions
- Clean resource management with try-finally blocks

2. Data Models (model.py):

The models define the database schema using SQLAlchemy ORM:

User Model:

- Stores all user account information

- Relationships to posts and comments
- Password stored as hash, never plain text

Post Model:

- Contains post content and metadata
- Foreign keys to user and subspace
- Timestamps for chronological ordering
- Relationships to comments and votes

Comment Model:

- Supports nested threading via parent_id
- Self-referential foreign key for replies
- Tracks post association and author

Vote Model:

- Composite unique constraint prevents duplicate votes
- Value constrained to -1 or 1
- Links users to posts they've voted on

Subspace Model:

- Hierarchical structure via parent_id
- Department and level organization
- Can be extended for more granular organization

3. Pydantic Schemas (schema.py):

Schemas define the structure of data for API requests and responses:

Request Schemas:

- UserCreate: Registration data

- PostCreate: New post data
- CommentCreate: New comment data
- LoginRequest: Login credentials

Response Schemas:

- UserOut: Public user information (excludes password)
- PostOut: Post with nested user, subspace, and comments
- CommentOut: Comment with user information
- SubspaceResponse: Subspace details

Benefits:

- Automatic validation of incoming data
- Type safety and IDE autocomplete
- Automatic API documentation generation
- Clear contracts between frontend and backend

4. Authentication System (auth.py):

Implements secure authentication using industry best practices:

Password Hashing:

- Uses bcrypt algorithm with automatic salt generation
- Computationally expensive to resist brute-force attacks
- One-way hashing (passwords cannot be decrypted)

JWT Token Authentication:

- Access tokens expire after 24 hours
- Tokens contain user identification in the payload
- Signed with secret key to prevent tampering

- Stateless authentication (no server-side session storage)

Security Considerations:

- Secret key stored as environment variable
- Password validation before token issuance
- Token verification on protected routes

5. API Routes:

Authentication Routes (auth_routes.py):

- POST /auth/signup: User registration
- POST /auth/login: User authentication and token generation

Post Routes (main.py):

- GET /posts: Retrieve all posts with related data
- POST /posts: Create new post
- GET /posts/{post_id}/comments: Get comments for a post
- POST /posts/{post_id}/comments: Add comment to post

Search Routes:

- GET /search-posts: Full-text search across posts

Subspace Routes:

- GET /subspace: List all subspaces
- POST /subspace: Create new subspace (admin feature)

6. WebSocket Implementation:

Real-time updates implemented using WebSocket protocol:

Connection Manager:

- Maintains list of active WebSocket connections

- Handles connection lifecycle (connect, disconnect)
- Broadcasts messages to all connected clients

Broadcasting Events:

- New posts trigger broadcast to all clients
- New comments trigger updates
- Vote changes broadcast score updates
- Clients receive JSON messages with event type and data

Benefits:

- Instant updates without page refresh
- Reduced server load compared to polling
- Better user experience with live content

7. CORS Configuration:

Cross-Origin Resource Sharing middleware allows web frontend to communicate with backend:

```
app.add_middleware(
    CORSMiddleware,
    allow_origins=["http://localhost:5175"],  # Vite dev server
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)
```

4.4.2 Web Frontend Implementation (React)

Project Structure:

Path	Description
bentalk-web/	This is the root directory of the React frontend prototype built with Vite.
├── src/	Contains all React source code files.
│ ├── App.jsx	Main React application component.
│ ├── index.css	Global CSS styles for the app.
│ └── main.jsx	Application entry point that renders React into the DOM.
├── public/	Static assets such as images, icons, and metadata.
├── package.json	Lists project dependencies and scripts.
└── vite.config.js	Vite configuration file for development and build settings.

Implementation Approach:

The web version uses React with Dexie.js for local data storage, creating a progressive web app experience. The implementation is contained primarily in a single comprehensive component for simplicity.

Key Features Implemented:

1. Local Database (Dexie.js):

- Browser-based IndexedDB storage
- Persistent across sessions
- Stores users, posts, comments, and votes locally
- Enables offline reading capabilities

2. Routing System:

4.4.2.1 Web Frontend Routing System (React Router)

Route Path	Page / Component	Description
/signup	SignUp Page	Entry point for new user registration.
/login	Login Page	User authentication screen for existing users.
/home	Department Listing	Displays list of all available departments.
/department/:dept	Level Selection	Shows available levels within the selected department.
/department/:dept/:level	Posts Listing	Displays posts related to the selected level.
/department/:dept/:level/post/:postId	Post Detail	Shows detailed view of a specific post with comments.

3. Authentication Flow:

- User signs up with academic credentials
- Credentials stored in localStorage for session persistence
- Protected routes redirect unauthenticated users to signup
- Logout clears session and returns to signup

4. Department Organization:

Seven departments from Faculty of Computing:

- Computer Science
- Cyber Security

- Data Science
- Software Engineering
- Information and Communication Technology
- Information Technology
- Information Science

Each department has four levels: 100L, 200L, 300L, 400L

5. Post Management:

- Create posts with title and content
- Posts automatically tagged with department, level, author, and timestamp
- Posts displayed newest-first by default
- Vote counts displayed with each post
- Click post to view full details and comments

6. Comment System:

- Nested comment threading (replies to replies)
- Tree-building algorithm organizes comments hierarchically
- Each comment shows author and timestamp
- Reply functionality creates parent-child relationships
- Delete functionality for own comments

7. Voting Mechanism:

- Upvote/downvote buttons on each post
- Visual indication of user's current vote
- Aggregate score displayed prominently
- Toggle vote on/off by clicking same button again

- Vote changes update immediately

8. Responsive Design:

- Desktop-optimized layout
- Mobile-responsive with flexbox and grid
- Touch-friendly interface elements
- Readable typography at all screen sizes

9. Placeholder Data Seeding:

On first load, the system creates placeholder posts for each department-level combination to demonstrate structure and provide starting content for testing.

4.4.3 Mobile Application Implementation (Kotlin/Android)

Project Structure:

Component / File	Purpose	Description
app/src/main/	Main Source Directory	Contains Kotlin source code, resources, and configuration files.
├── MainActivity.kt	App Entry Point	Initial activity launched when the app starts.
├── models/	Data Models	Defines data classes representing app entities. These match backend schema's received as JSON
│ ├── User.kt	User Model	Represents user data and profile details.
│ ├── Post.kt	Post Model	Represents user-created posts.
│ ├── Comment.kt	Comment Model	Represents comments associated with posts.
│ ├── Subspace.kt	Subspace Model	Represents community or topic grouping.
│ └── LoginResponse.kt	Response Model	Handles API login response data.
├── network/	Networking Layer	Handles communication with the backend via API and WebSocket.
│ ├── ApiService.kt	API Endpoints	Defines Retrofit endpoints for HTTP requests.
│ ├── RetrofitClient.kt	Retrofit Configuration	Initializes Retrofit client and base URL.

│ └── LiveSocketClient.kt	WebSocket Client	Manages real-time communication with the server.
├── screens/	UI Screens	Contains composable screens for app navigation and interaction.
│ ├── HomeScreen.kt	Home Screen	Displays main feed and posts.
│ ├── LoginScreen.kt	Login Screen	User authentication interface.
│ ├── PostsScreen.kt	Posts Screen	Displays list of posts in a subspace.
│ ├── CreatePostScreen.kt	Create Post Screen	Form to create new posts.
│ ├── SubspacesScreen.kt	Subspaces Screen	Lists all available subspaces.
│ ├── CreateSubspaceScreen.kt	Create Subspace	Allows user to create new community spaces.
│ └── ProfileScreen.kt	Profile Screen	Displays and edits user information.
├── navigation/	App Navigation	Handles navigation between composable screens.
│ └── Routes.kt	Routes Definition	Defines navigation paths and routes constants.
├── ui/theme/	Theming	Holds UI theme definitions and color schemes.
│ └── Theme.kt	Theme File	Defines typography, color palette, and app theme.
├── res/	Resources	Contains drawable assets, layouts, and string values.
│ ├── values/	Values Folder	Holds XML files for strings, dimensions, and styles.
│ └── drawable/	Drawable Folder	Stores image and vector assets used by UI.
└── AndroidManifest.xml	Manifest File	Declares app permissions, components, and configurations.

Implementation Details:

1. Network Layer (Retrofit):

RetrofitClient.kt:

- Configures base URL for API communication
- Automatically detects emulator vs. real device

- Uses appropriate localhost address (10.0.2.2 for emulator)
- Converts JSON responses to Kotlin objects using Gson

ApiService.kt:

- Interface defining all API endpoints
- Type-safe method calls
- Automatic serialization/deserialization
- Support for path parameters, query parameters, and request bodies

2. Data Models:

Kotlin data classes mirror backend schemas:

- Automatic JSON parsing
- Null safety with Kotlin's type system
- Immutable by default
- Concise syntax

3. UI Implementation (Jetpack Compose):

Modern declarative UI framework:

Composable Functions:

- Each screen is a composable function
- State-driven rendering
- Automatic recomposition on state changes

Material Design 3:

- Modern visual design
- Consistent with Android platform guidelines
- Adaptive color schemes

- Accessible components

4. Navigation:

Navigation Compose handles screen transitions:

- Type-safe navigation arguments
- Back stack management
- Deep linking support
- Transition animations

5. State Management:

State Hoisting:

- State lifted to appropriate level
- Unidirectional data flow
- Predictable state updates

Remember State:

- Survives recomposition
- MutableState for reactive updates

6. Key Screens:

LoginScreen:

- Email/username/matric ID identifier
- Password input with visual transformation
- Loading indicator during authentication
- Error message display
- Navigation to home on success

HomeScreen:

- Navigation hub to main features
- Quick access buttons to posts, subspaces, profile
- Create content shortcuts

PostsScreen:

- LazyColumn for efficient scrolling
- Post cards with expand/collapse comments
- Inline comment input
- Real-time comment addition
- Loading and error states

CreatePostScreen:

- Title and content text fields
- Form validation
- API call with loading state
- Success feedback and navigation

7. Networking and Async Operations:

Kotlin Coroutines:

- Asynchronous API calls
- Main-safe operations
- Structured concurrency
- Exception handling

Retrofit Callbacks:

- onResponse: Handle successful API responses

- onFailure: Handle network errors
- Update UI on main thread

8. WebSocket Integration:

LiveSocketClient:

- Persistent connection to backend
- Real-time event reception
- Automatic reconnection on disconnect
- Event parsing and handling

9. Local Data Persistence:

SharedPreferences:

- Store user session data
- Persist login state
- Cache user profile information
- Quick access to frequently used data

4.5 System Testing

4.5.1 Unit Testing

Backend Unit Tests:

- Password hashing verification
- JWT token generation and validation
- Database model relationships
- Pydantic schema validation

Test Framework: pytest

Coverage: Core authentication and data validation functions

4.5.2 Integration Testing

API Endpoint Testing:

- User registration flow
- Login and token retrieval
- Post creation and retrieval
- Comment addition
- Vote casting
- Search functionality

Tools: Postman, curl, Python requests library

Test Scenarios:

- Valid inputs produce expected responses
- Invalid inputs return appropriate error codes
- Authentication required for protected endpoints
- Foreign key constraints prevent orphaned records

4.5.3 User Interface Testing

Web Application:

- Cross-browser testing (Chrome, Firefox, Safari, Edge)
- Responsive design verification at different screen sizes
- Form validation feedback
- Navigation flow testing
- Real-time update verification

Mobile Application:

- Emulator testing across different API levels
- Screen orientation changes

- Network interruption handling
- Memory leak detection

4.5.4 Performance Testing

Backend Performance:

- API response times under load
- Database query optimization
- WebSocket connection scaling

Frontend Performance:

- Page load times
- Time to interactive
- Bundle size optimization
- Rendering performance for large lists

Tools: Browser DevTools and Lighthouse (Chrome network metrics)

4.5.5 Security Testing

Authentication Security:

- Password strength requirements
- Token expiration enforcement
- SQL injection prevention
- XSS attack prevention

Data Protection:

- Password hashing verification
- Secure token transmission
- Input sanitization

4.5.6 Usability Testing

Methodology:

- Think-aloud protocol with 5-7 users
- Task completion observation
- Post-test questionnaires
- Iterative design improvements

Tasks Tested:

- Account creation
- Finding specific department/level
- Creating a post
- Adding a comment
- Voting on posts

Metrics:

- Task completion rate
- Time to complete tasks
- Error frequency

4.6 Challenges Encountered

4.6.1 Technical Challenges

1. WebSocket Connection Management:

Challenge: Maintaining stable WebSocket connections across different network conditions and handling reconnection scenarios.

Solution: Implemented connection monitoring with automatic reconnection logic. Added heartbeat mechanism to detect stale connections.

2. Mobile Network Configuration:

Challenge: Android emulator and physical device require different base URLs for localhost communication.

Solution: Created detection logic in RetrofitClient to automatically select appropriate URL based on device type (emulator uses 10.0.2.2, physical devices use WiFi IP).

3. Nested Comment Threading:

Challenge: Building and rendering tree structure for nested comments efficiently.

Solution: Developed recursive tree-building algorithm that constructs hierarchical structure from flat comment list. Implemented depth-limited rendering to prevent excessive nesting.

4. Real-time State Synchronization:

Challenge: Keeping multiple clients synchronized when posts, comments, or votes change.

Solution: WebSocket broadcasts for all mutating operations. Clients update local state immediately (optimistic updates) then sync with broadcast confirmation.

5. Database Query Optimization:

Challenge: N+1 query problem when loading posts with related users, subspaces, and comments.

Solution: Used SQLAlchemy's joinedload to eager-load relationships in single query. Added appropriate indexes on foreign keys.

4.6.2 Development Environment Challenges

1. CORS Issues:

Challenge: Browser blocking API requests due to cross-origin restrictions during development.

Solution: Configured CORS middleware in FastAPI to allow requests from Vite dev server. Set appropriate origins, methods, and headers.

2. PostgreSQL Setup:

Challenge: Initial database configuration and connection issues on development machine.

Solution: Used Docker container for PostgreSQL to ensure consistent environment. Created initialization scripts for database setup.

3. Android Studio Configuration:

Challenge: Kotlin and Jetpack Compose were new technologies requiring setup and configuration.

Solution: Followed official Android documentation. Used gradle dependency management for consistent builds.

4.6.3 Design Challenges

1. Information Architecture:

Challenge: Organizing departments and levels in intuitive hierarchy without overwhelming users.

Solution: Implemented three-level navigation: Departments → Levels → Posts. Breadcrumb navigation shows current location.

2. Mobile UI Constraints:

Challenge: Displaying nested comments and long posts on small screens without excessive scrolling.

Solution: Implemented expand/collapse functionality. Limited initial comment depth. Added "show more" interactions for long content.

3. Balancing Features vs. Simplicity:

Challenge: Including necessary features without creating overwhelming interface.

Solution: Focused on core features for MVP: posts, comments, votes, search. Planned additional features (moderation, notifications) for future iterations.

4.6.4 Time and Resource Constraints

1. Solo Development:

Challenge: Implementing both backend and two frontend platforms as single developer.

Solution: Prioritized features by importance. Reused patterns across platforms. Leveraged frameworks to avoid reinventing solutions.

2. Testing Coverage:

Challenge: Limited time for comprehensive testing across all features and platforms.

Solution: Focused testing on critical paths (authentication, post creation, comment threading). Automated where possible. Documented known issues for future resolution.

3. Documentation:

Challenge: Maintaining documentation while rapidly developing features.

Solution: Inline code comments for complex logic. README files for setup instructions. API documentation auto-generated by FastAPI.

CHAPTER FIVE

SUMMARY, CONCLUSION AND RECOMMENDATIONS

5.1 Summary

This project successfully designed and implemented BenTalk, a centralized discussion platform specifically tailored for the University of Benin's Faculty of Computing. The system addresses the critical challenge of fragmented academic communication by providing a structured, dedicated space where students can engage in knowledge sharing, peer-to-peer learning, and collaborative problem-solving.

Key Accomplishments:

1. Comprehensive System Development:

- Fully functional backend API built with Python FastAPI
- PostgreSQL database with normalized schema
- Responsive web application using React
- Native Android mobile application using Kotlin and Jetpack Compose
- Real-time updates via WebSocket integration

2. Core Features Implemented:

- User registration and authentication with JWT tokens
- Department and level-based subspace organization
- Post creation with title and content
- Nested comment threading for discussions
- Upvote/downvote system for content quality signaling
- Full-text search across posts
- Real-time updates for new content

3. Academic-Focused Design:

- Structure aligned with university hierarchy (departments and levels)
- Knowledge preservation through permanent, searchable archives
- Balance between formal academic structure and informal peer interaction
- Mobile-first design accommodating Nigerian infrastructure constraints

4. Technical Achievements:

- Cross-platform accessibility (web and mobile)
- Efficient database design with proper indexing
- Secure authentication and password hashing
- RESTful API architecture with comprehensive endpoints
- Modern UI frameworks (React, Jetpack Compose)

5. Research Contributions:

- Thorough literature review establishing theoretical foundations
- Gap analysis identifying specific needs in Nigerian universities
- Proof-of-concept demonstrating viability of dedicated academic platforms
- Documentation providing blueprint for similar implementations

The system successfully demonstrates that purpose-built platforms can address the limitations of general social media for academic use while maintaining the ease of use and accessibility that makes social media attractive to students.

5.2 Conclusion

The development of BenTalk validates the central hypothesis that a department-based academic discussion platform can significantly enhance intra-faculty collaboration and knowledge sharing among students and lecturers. The project addresses real problems faced by Nigerian university

students who currently rely on fragmented, informal communication channels for academic purposes.

Key Findings:

1. Technical Feasibility: The project demonstrates that modern web and mobile technologies can be leveraged to create sophisticated academic platforms at minimal cost. Open-source frameworks (FastAPI, React, Kotlin) provide robust foundations for building scalable systems without expensive licensing fees.

2. Structural Alignment: Organizing discussions by department and academic level proves to be an intuitive model that aligns with how students already conceptualize their academic communities. This structure provides natural boundaries that reduce noise while maintaining relevance.

3. Knowledge Preservation Value: Unlike ephemeral chat messages, permanent discussion archives create lasting value. Students can search past discussions, reducing redundant questions and building collective institutional knowledge that benefits future cohorts.

4. Platform-Specific Advantages: Purpose-built platforms can incorporate academic-specific features (departmental organization, formal/informal balance, knowledge preservation) that general social media platforms cannot provide while still maintaining the ease of use students expect.

5. Scalability Potential: While developed for the Faculty of Computing, the architecture and approach are generalizable. The same model could be extended to other faculties, other universities, or even adapted for different educational contexts.

Limitations Acknowledged:

1. Prototype Status: As a development-stage prototype, the system has not undergone extensive field testing with large numbers of users. Real-world usage patterns may reveal unforeseen challenges or necessary feature adjustments.

2. Limited Feature Set: The current implementation focuses on core functionality. Advanced features like content moderation tools, notification systems, direct messaging, and analytics dashboards remain to be implemented.

3. Infrastructure Dependencies: The system requires reliable internet connectivity and assumes users have access to smartphones or computers. While designed to be lightweight, it still faces the same infrastructure challenges as other online platforms in Nigeria.

4. Adoption Challenge: The success of any platform depends on achieving critical mass of users. Without institutional support and active user recruitment, even well-designed platforms may struggle to gain traction.

5. Maintenance Requirements: Sustaining the platform requires ongoing technical maintenance, server hosting costs, and responsive support for user issues resources that may not be readily available without institutional backing.

Despite these limitations, BenTalk represents a significant step toward solving the real problem of fragmented academic communication in Nigerian universities. The project demonstrates that locally-developed, context-appropriate solutions can effectively address challenges that generic, global platforms do not.

5.3 Recommendations

Based on the development experience and evaluation of BenTalk, the following recommendations are made for stakeholders:

5.3.1 For University Administration

1. Institutional Adoption: The University of Benin should consider piloting BenTalk within the Faculty of Computing. Institutional support would provide:

- Official backing encouraging student participation
- Resources for hosting and maintenance
- Integration with existing university systems (authentication, student records)
- Legitimacy that increases user trust and adoption

2. Faculty Involvement: Encourage faculty members to actively participate in the platform by:

- Posting official announcements in relevant subspaces
- Answering student questions to provide authoritative information
- Moderating discussions to ensure accuracy and appropriateness
- Recognizing valuable student contributions

3. Resource Allocation: Provide technical resources for:

- Server hosting in reliable data centers
- Regular backups and security audits
- Technical support staff to handle user issues
- Continuous development and feature enhancement

4. Policy Development: Establish clear policies regarding:

- Acceptable use guidelines
- Privacy and data protection
- Content moderation standards
- Consequences for violations
- Faculty participation expectations

5.3.2 For Developers and Technical Teams

1. Feature Enhancements:

High Priority:

- Notification system (email and push notifications)
- Advanced search with filters (date range, author, department)
- User reputation system recognizing helpful contributors
- Content moderation dashboard for faculty
- Analytics dashboard showing usage patterns and popular topics

Medium Priority:

- Direct messaging between users
- File attachment support for posts and comments
- Pinned posts for important announcements
- Post editing history and version control
- Mobile app for iOS

Future Considerations:

- AI-powered content recommendations
- Automatic question-answer matching
- Sentiment analysis for community health monitoring
- Integration with university learning management systems
- Multi-language support (English, Pidgin, local languages)

2. Performance Optimization:

- Implement caching layer (Redis) for frequently accessed data
- Add CDN for static asset delivery
- Optimize database queries with query profiling

- Implement pagination for large result sets
- Add lazy loading for images and media content

3. Security Hardening:

- Implement two-factor authentication
- Add rate limiting to prevent abuse
- Conduct regular security audits
- Implement comprehensive logging and monitoring
- Add CAPTCHA for registration to prevent bots

4. Scalability Improvements:

- Containerize application with Docker
- Implement horizontal scaling with load balancers
- Add database replication for read scaling
- Implement microservices architecture for larger deployments
- Add message queue for background tasks

5. Code Quality:

- Increase unit test coverage to 80%+
- Add integration tests for critical workflows
- Implement continuous integration/continuous deployment (CI/CD)
- Conduct regular code reviews
- Maintain comprehensive documentation

5.3.3 For Students and End Users

1. Active Participation: Students should embrace the platform by:

- Asking thoughtful questions rather than relying solely on WhatsApp

- Providing detailed, helpful answers to peer questions
- Upvoting quality content to help others find valuable information
- Reporting inappropriate content or misinformation
- Sharing the platform with classmates to grow the community

2. Content Quality:

- Write clear, well-formatted posts with descriptive titles
- Search before posting to avoid duplicate questions
- Provide context when asking questions (course, topic, what you've tried)
- Edit posts to add solutions when problems are resolved
- Give credit and cite sources when sharing information from elsewhere

3. Community Standards:

- Maintain respectful, professional discourse
- Help newcomers learn how to use the platform effectively
- Avoid off-topic conversations in academic subspaces
- Protect privacy by not sharing personal contact information
- Report bugs and suggest improvements to help platform evolve

5.3.4 For Other Universities

1. Adaptation Framework: Other institutions can adapt BenTalk by:

- Customizing department and level structures to match their organization
- Adjusting branding and visual design to reflect institutional identity
- Modifying authentication to integrate with existing systems
- Adding institution-specific features (course codes, academic calendar)
- Localizing language and terminology

2. Implementation Strategy:

- Start with pilot program in single faculty
- Gather feedback and iterate on design
- Gradually expand to other faculties
- Provide training for students and faculty
- Monitor adoption metrics and adjust strategy

3. Sustainability Planning:

- Secure funding for ongoing hosting and maintenance
- Establish technical support team
- Create student ambassador program to promote adoption
- Develop content moderation guidelines and team
- Plan for regular feature updates and improvements
-

5.4 Final Thoughts

BenTalk represents more than just a software application; it embodies a vision of how technology can be thoughtfully applied to address real educational challenges in the Nigerian context. While global platforms like Facebook and WhatsApp have been adapted for academic use, their fundamental design remains misaligned with educational needs. Purpose-built platforms like BenTalk can provide the structure, focus, and institutional integration that academic communities require.

The project demonstrates that local developers, with understanding of contextual challenges and constraints, can create solutions superior to globally-designed platforms for their specific contexts. Nigerian universities face unique challenges: infrastructure limitations, resource

constraints, specific organizational structures that require tailored approaches rather than one-size-fits-all solutions.

However, technology alone cannot solve all problems. The success of platforms like BenTalk depends equally on:

- **Institutional support**: Universities must recognize, endorse, and resource such initiatives
- **Faculty involvement**: Lecturers must see value and actively participate
- **Student adoption**: Users must choose the platform over existing alternatives
- **Community building**: Norms of quality discourse must be established and maintained
- **Continuous improvement**: Platforms must evolve based on user feedback and changing needs

As Nigerian universities continue their digital transformation journey, projects like BenTalk offer valuable lessons. They show that effective educational technology requires understanding pedagogy, respecting institutional structures, considering infrastructure realities, and centering user needs in design processes.

The future of academic communication likely lies not in forcing academic purposes onto social platforms, nor in rigid, bureaucratic learning management systems, but in purpose-built spaces that combine the best of both: the ease and engagement of social media with the structure and focus of academic institutions.

This project is offered as a contribution to that future a working prototype, a methodological approach, and an invitation to continue building better tools for African higher education.

REFERENCES

Adewale, T. & Aremu, A. (2020). *E-learning adoption in Nigerian universities: Challenges and opportunities*. Journal of Education and Information Technologies, 25(3), 1561–1577.

Al Balushi, W., Al-Busaidi, F. S., Malik, A., & Al-Salti, Z. (2022). Social media use in higher education during the COVID-19 pandemic: A systematic literature review. *International Journal of Emerging Technologies in Learning (iJET), 17*(24), 4–24. https://doi.org/10.3991/ijet.v17i24.32399

Bouhnik, D., & Deshen, M. (2014). WhatsApp goes to school: Mobile instant messaging between teachers and students. *Journal of Information Technology Education: Research, 13*, 217-231.

Boyd, D. M., & Ellison, N. B. (2007). Social network sites: Definition, history, and scholarship. *Journal of Computer-Mediated Communication, 13*(1), 210-230.

Hauben, M., & Hauben, R. (1997). The social forces behind the development of Usenet. In *Netizens: On the history and impact of Usenet and the Internet*. IEEE Computer Society Press. https://www.columbia.edu/~hauben/book-pdf/CHAPTER%203.pdf

Jones, T., & Cuthrell, K. (2011). YouTube: Educational potentials and pitfalls. *Computers in the Schools, 28*(1), 75-85.

Junco, R. (2012). The relationship between frequency of Facebook use, participation in Facebook activities, and student engagement. *Computers & Education, 58*(1), 162-171.

Kirschner, P. A., & Karpinski, A. C. (2010). Facebook and academic performance. *Computers in Human Behavior, 26*(6), 1237-1245.

LaFlamme, K. A. (2025). *Leveraging Reddit in academia*. Discover Education, 4, 163. https://doi.org/10.1007/s44217-025-00542-2

Chugh, R., Grose, R., & Macht, S. A. (2020). *Social media usage by higher education academics: A scoping review of the literature. Education and Information Technologies, 26*(1), 983–1006. https://doi.org/10.1007/s10639-020-10288-z

Rexwhite, T. E. & Fasae, J. K. (2022). *WhatsApp application for teaching and learning in higher education institutions*. ResearchGate. https://www.researchgate.net/profile/Nnnnnnnn-Nnnnnnnnn/publication/364511337_WhatsApp_application_for_teaching_and_learning_in_higher_education_institutions/links/6368116737878b3e878a2747/WhatsApp-application-for-teaching-and-learning-in-higher-education-institutions.pdf

Roblyer, M. D., McDaniel, M., Webb, M., Herman, J., & Witty, J. V. (2010). Findings on Facebook in higher education: A comparison of college faculty and student uses and perceptions of social networking sites. *Internet and Higher Education, 13*(3), 134-140.

Securities and Exchange Commission. (2022). Meta Platforms, Inc. Form 10-K Annual Report. Retrieved from https://investor.fb.com/financials/

Siemens, G. (2005). Connectivism: A learning theory for the digital age. *International Journal of Instructional Technology and Distance Learning, 2*(1), 3-10.

Sventek, V. A. (1984, October). *A tutorial for the Software Tools Mail System MSG* (PUB-3049). Lawrence Berkeley National Laboratory, University of California. https://escholarship.org/uc/item/0kc9c76c

Tan, E., & Pearce, N. (2011). Open education videos in the classroom: Exploring the opportunities and barriers to the use of YouTube in teaching introductory sociology. *Research in Learning Technology, 19*(Supplement), 125-132.

Tess, P. A. (2013). The role of social media in higher education classes (real and virtual) – A literature review. *Computers in Human Behavior, 29*(5), A60–A68.

Van Den Beemt, A., Thurlings, M., & Willems, M. (2020). Towards an understanding of social media use in the classroom: A literature review. *Technology, Pedagogy and Education, 29*(1), 35–55. https://doi.org/10.1080/1475939X.2019.1695657

Wenger, E. (1998). *Communities of practice: Learning, meaning, and identity*. Cambridge University Press.

Woolley, D. R. (2016). *PLATO: The emergence of online community.* In J. Malloy (Ed.), *Social media archeology and poetics* (pp. 103-118). The MIT Press.

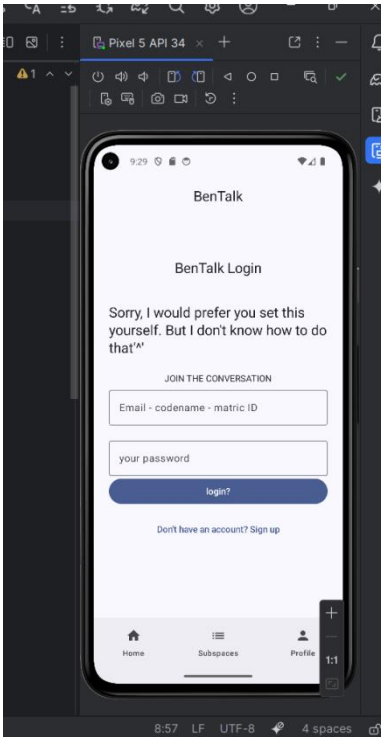
Zachos, G., Paraskevopoulou-Kollia, E.-A., & Anagnostopoulos, I. (2018). *Social media use in higher education: A review. Education Sciences, 8*(4), 194. https://doi.org/10.3390/educsci8040194

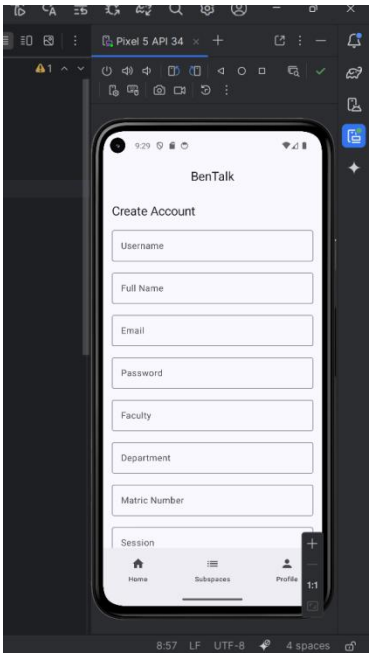
Zhao, D., & Rosson, M. B. (2009). How and why people Twitter: The role that micro-blogging plays in informal communication at work. *Proceedings of the ACM 2009 International Conference on Supporting Group Work*, 243–252.

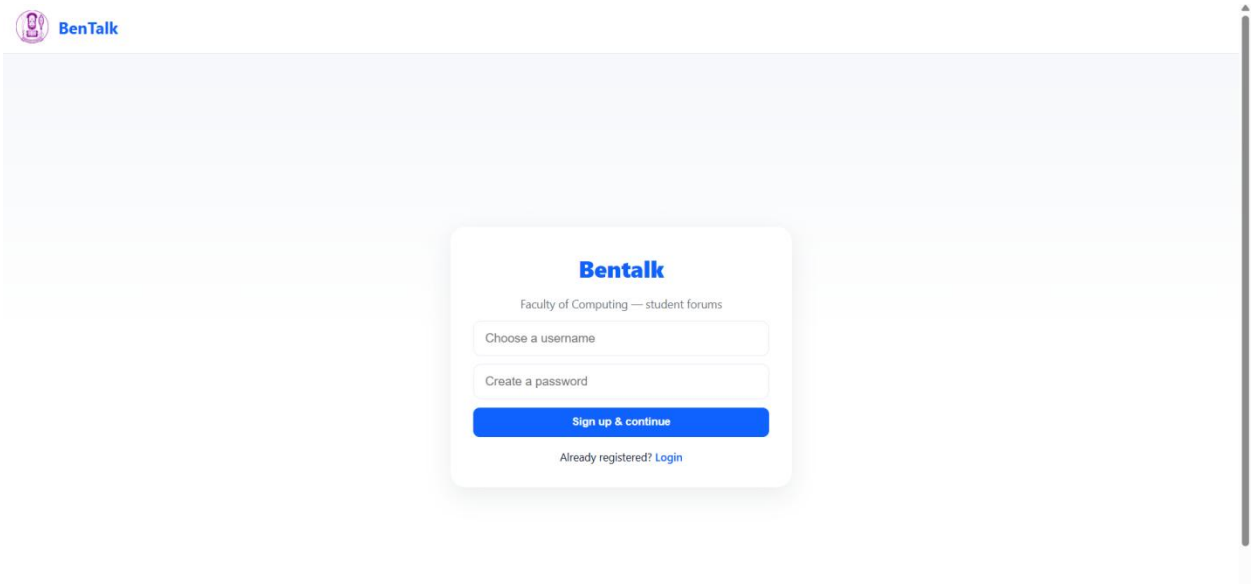
Zide, J., Elman, B., & Shahani-Denning, C. (2014). LinkedIn and recruitment: How profiles differ across occupations. *Employee Relations, 36*(5), 583-604.

Appendix A: System Screenshots

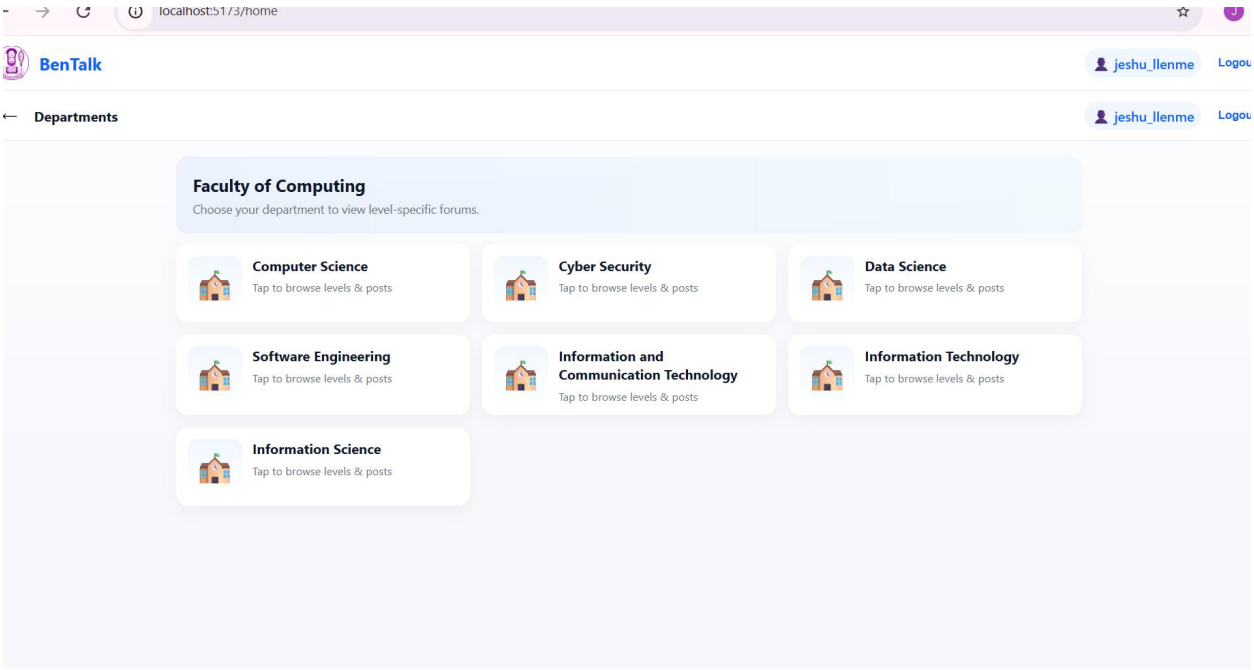
1. Login/Signup screens (web and mobile)



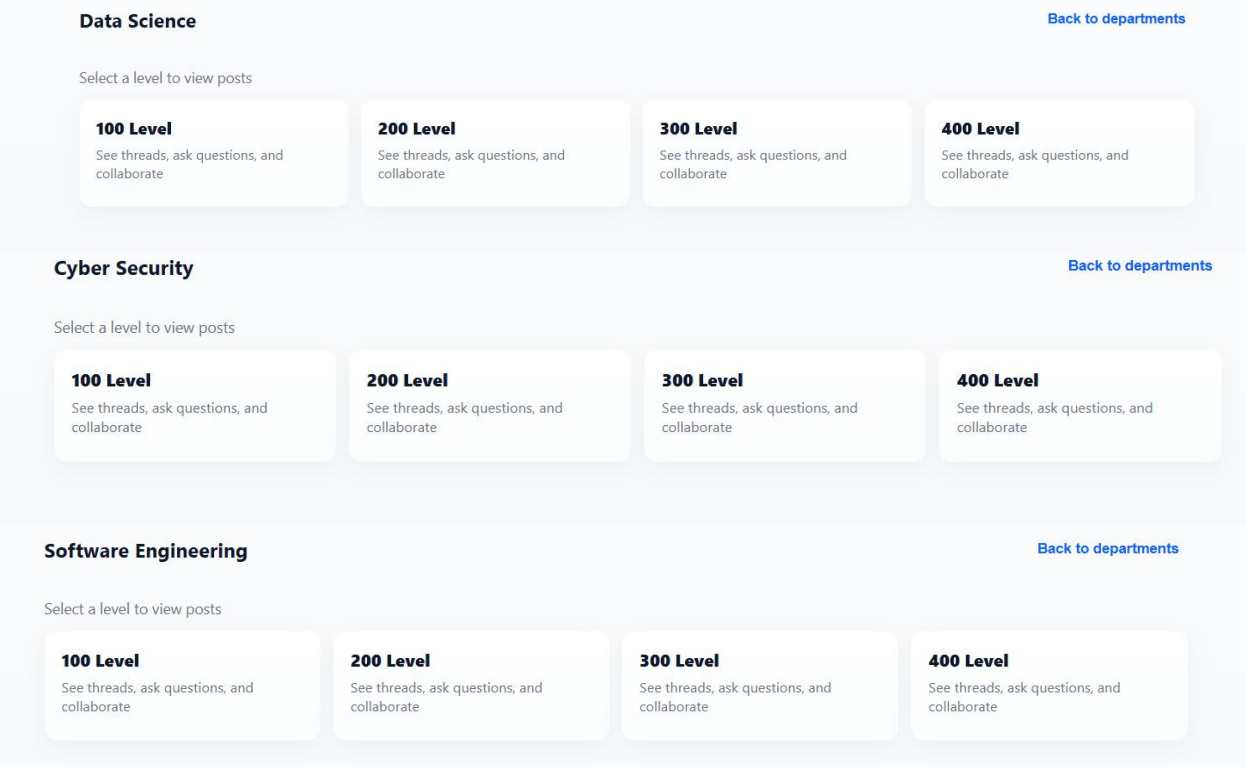




2. Department listing page

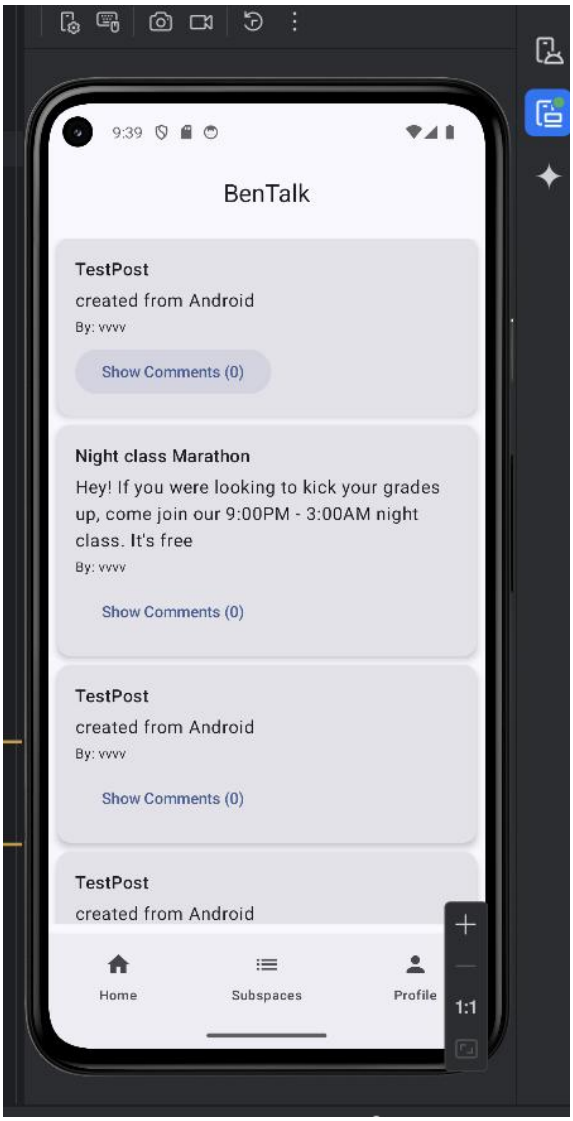


3. Level selection page



4. Posts listing with vote counts

5. Post detail with comments



welcome to nigner ed I guess.

Back Back to threads

Software Engineering • 100 Level • By jeshu_llenme • 11/10/2025, 1:02:48 PM

Sometimes I wonder if the lecturers realize how much time their assignments require. It feels like they assume we all have endless hours and perfect laptops. I'm trying my best, but it's exhausting.

1

Comments

Add a comment...

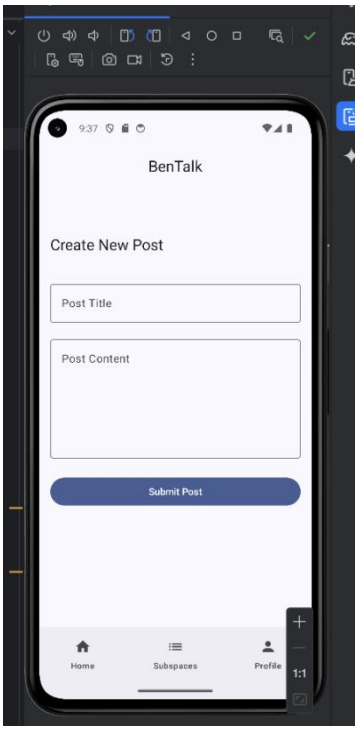
Comment

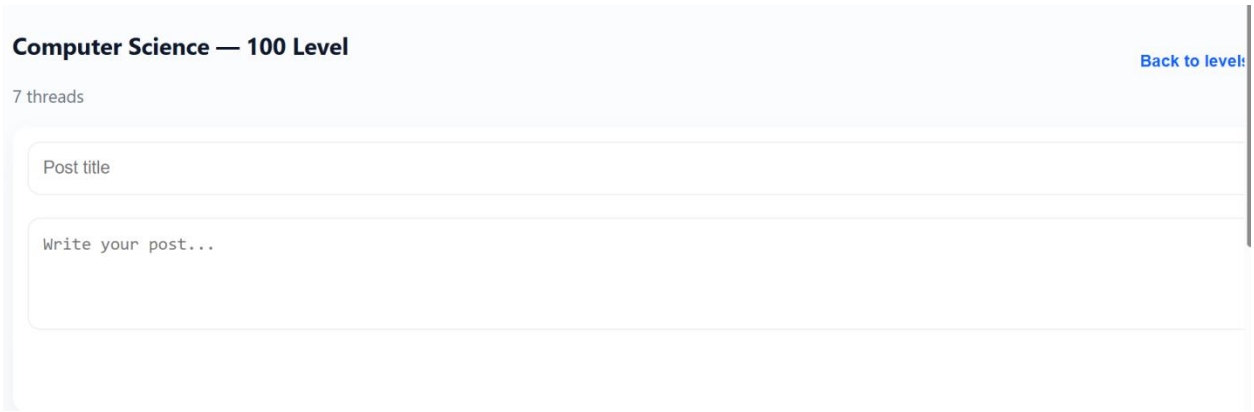
User #6 11/11/2025, 9:52:17 AM

guy rest

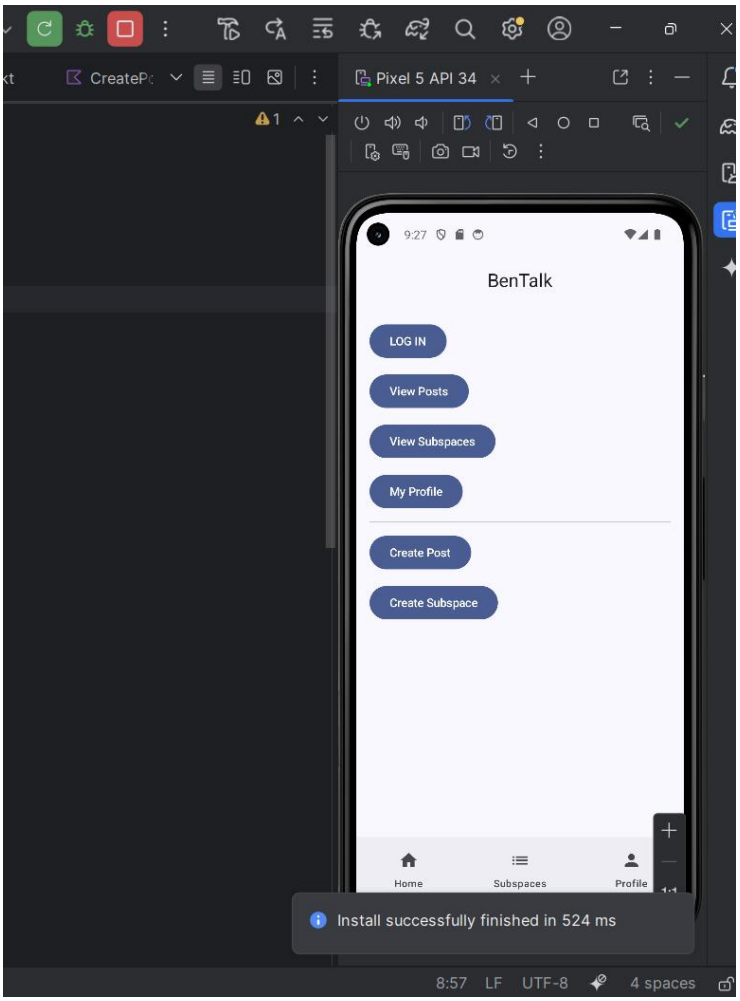
Reply Delete

6. Create post interface

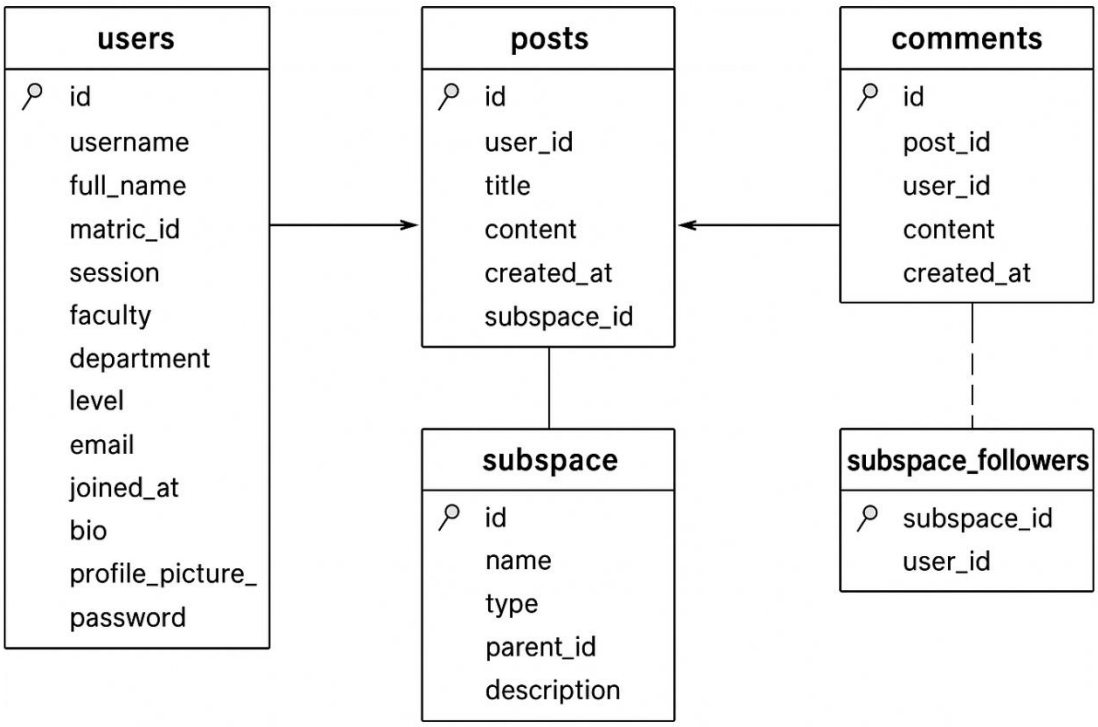




7. Mobile app screens showing key features



8. Database schema diagram


users
id
username
full_name
matric_id
session
faculty
department
level
email
joined_at
bio
profile_picture_
password
posts
id
user_id
title
content
created_at
subspace_id
comments
id
post_id
user_id
content
created_at
subspace
id
name
type
parent_id
description
subspace_followers
subspace_id
user_id

Appendix B: Code Samples

1. FastAPI route implementation

```
router= APIRouter()

@router.post("/signup", response_model = schema.UserOut)

def sign_up(user: schema.UserCreate, db: Session = Depends(get_db)):

print("entered signup")

existing_user = db.query(model.User).filter((model.User.username == user.username) |
(model.User.email == user.email)).first()

if existing_user:

raise HTTPException(status_code= 400, detail= "Username already exists")

hash_pwd = hash_password(user.password)

db_user = model.User(
username = user.username,
full_name = user.full_name,
matric_id = user.matric_id,
faculty= user.faculty,
department= user.department,
password= hash_pwd,
bio= getattr(user, "bio", None),
profile_picture_url= getattr(user, "profile_picture_url", None),
session= user.session,
level= user.level,
email= user.email
)
db.add(db_user)
db.commit()
db.refresh(db_user)
return db_user
@router.post("/login")
def login( request: schema.LoginRequest, db: Session = Depends(get_db)):
print("entered login")
print("Login attempt:", request.identifier)
db_user = (db.query(model.User).filter((model.User.username == request.identifier)
|(model.User.email == request.identifier) | (model.User.matric_id == request.identifier)).first())
print(db_user)
if not db_user:
raise HTTPException(status_code= 400, detail= "incorrect username or password")
```

```
if not verify_password(request.password, db_user.password):
raise HTTPException(status_code= 400, detail ="incorrect username or password")
access_token_expires = timedelta(minutes= ACCESS_TOKEN_EXPIRE_MINUTES)
access_token = create_access_token(
data= {"sub": db_user.username}, expires_delta= access_token_expires
)
return {
"access_token": access_token,
"token_type": "bearer",
"user": {
"id": db_user.id,
"username": db_user.username,
"email": db_user.email,
"matric_id": db_user.matric_id,
},
}
@app.post("/posts/{post_id}/comments", response_model=schema.CommentOut)
async def create_comment(
post_id: int, comment: schema.CommentCreate, db: Session = Depends(get_db)
):
db_post = db.query(model.Post).filter(model.Post.id == post_id).first()
if not db_post:
raise HTTPException(status_code=404, detail="Post not found")
db_user = db.query(model.User).filter(model.User.id == comment.user_id).first()
if not db_user:
raise HTTPException(status_code=404, detail="User not found")
db_comment = model.Comment(
post_id=post_id, user_id=comment.user_id, content=comment.content
)
db.add(db_comment)
db.commit()
db.refresh(db_comment)
await manager.broadcast({
"event": "new comment",
"data":schema.CommentOut.from_orm(db_comment).dict})
return db_comment
@app.get("/")
def read_root():
return {"message":"BenTalk backend is alive!"}
@app.get("/posts", response_model= List[schema.PostOut])
def get_posts(db: Session= Depends(get_db)):
posts= db.query(model.Post).options(
joinedload(model.Post.user),
joinedload(model.Post.subspace),
joinedload(model.Post.comments).joinedload(model.Comment.user)
).all()
```

```
return posts
@app.post("/posts", response_model= schema.PostOut)
async def create_post(post: schema.PostCreate,db: Session = Depends(get_db)):
db_user= db.query(model.User).filter(model.User.id == post.user_id).first()
if not db_user:
raise HTTPException(status_code=400, detail="User does not exist")
db_post= model.Post(
title= post.title,
content= post.content,
user_id= post.user_id,
)
db.add(db_post)
db.commit()
db.refresh(db_post)
await manager.broadcast({
"event": "new_post",
"data": schema.PostOut.from_orm(db_post).dict()
})
return db_post
@app.post("/users", response_model= schema.UserOut)
def create_user(user: schema.UserCreate, db: Session = Depends(get_db)):
db_user= model.User(**user.dict())
db.add(db_user)
db.commit()
db.refresh(db_user)
return db_user
@app.get("/subspace", response_model= List[SubspaceResponse])
def get_subspaces(db: Session = Depends(get_db)):
return db.query(model.Subspace).all()
@app.post("/subspace", response_model= SubspaceResponse)
def create_subspace(subspace: SubspaceCreate, db: Session= Depends(get_db)):
new_subspace= model.Subspace(
name= subspace.name,
type= subspace.type,
parent_id= subspace.parent_id,
description= subspace.description
)
db.add(new_subspace)
db.commit()
db.refresh(new_subspace)
return new_subspace
@app.get("/posts/{post_id}/comments", response_model=List[schema.CommentOut])
def get_comments(post_id: int, db: Session = Depends(get_db)):
comments = (
db.query(model.Comment)
.filter(model.Comment.post_id == post_id)
```

```
.options(joinedload(model.Comment.user))
.all()
)
return comments
```

2. SQLAlchemy model definitions

```
from sqlalchemy import Column, Integer, String, Text, ForeignKey, DateTime, Table, func
from sqlalchemy.dialects.postgresql import TSVECTOR
from sqlalchemy import Date,Time, String
from database import Base
from sqlalchemy.orm import relationship
class Post(Base):
__tablename__ = "posts"
id = Column(Integer, primary_key=True)
user_id = Column(Integer,ForeignKey("users.id"), nullable=False)
title = Column(String, nullable=False)
content = Column(Text, nullable=False)
created_at= Column(DateTime(timezone= True), server_default=func.now())
subspace_id= Column(Integer, ForeignKey("subspace.id"), nullable= True)
user= relationship("User", back_populates="posts")
subspace= relationship("Subspace", back_populates="posts")
comments= relationship("Comment", back_populates="post")
subspace_followers= Table(
"subspace_followers",
Base.metadata,
Column("subspace_id", Integer, ForeignKey("subspace.id")),
Column("user_id", Integer, ForeignKey("users.id"))
)
class User(Base):
__tablename__="users"
id = Column(Integer, primary_key=True, index=True)
username = Column(String, unique=True, nullable=False)
full_name = Column(String, nullable=False)
matric_id = Column(String, nullable= False, unique=True)
session = Column(String(22), nullable=False)
faculty = Column(String, nullable=False)
department = Column(String, nullable=False)
level = Column(String, nullable=False)
email= Column(String, unique=True, index= True, nullable= True)
joined_at= Column(DateTime(timezone=True), server_default=func.now())
bio= Column(Text, nullable= True)
profile_picture_url= Column(String, nullable= True)
password= Column(String, nullable = False)
posts= relationship("Post", back_populates="user")
comments= relationship("Comment", back_populates="user")
```

```
followed_subspaces= relationship("Subspace", back_populates="followers", secondary=
subspace_followers)
class Subspace(Base):
__tablename__="subspace"
id = Column(Integer, index= True, primary_key= True)
name = Column(String, unique= True,index= True, nullable= False)
type= Column(String,nullable= False)
parent_id= Column(Integer, ForeignKey("subspace.id"), nullable= True)
description= Column(Text, nullable= True)
created_at= Column(DateTime(timezone= True), server_default= func.now())
parent= relationship("Subspace", remote_side=[id])
posts= relationship("Post", back_populates="subspace")
followers= relationship( "User", secondary= subspace_followers, back_populates=
"followed_subspaces")
class Comment(Base):
__tablename__="comments"
id= Column(Integer, primary_key= True)
post_id= Column(Integer, ForeignKey("posts.id"), nullable= False)
user_id= Column(Integer, ForeignKey("users.id"), nullable= False)
content= Column(Text, nullable= False)
created_at= Column(DateTime(timezone= True), server_default= func.now())
post= relationship("Post", back_populates="comments")
user= relationship("User", back_populates="comments")
```

3. React component structure

```
<Routes>
<Route path="/" element={<Navigate to="/signup" replace />} />
<Route path="/signup" element={<SignUp />} />
<Route path="/login" element={<Login />} />
<Route path="/home" element={<ProtectedRoute><HomePage /></ProtectedRoute>} />
<Route path="/department/:dept" element={<ProtectedRoute><DepartmentLevels
/></ProtectedRoute>} />
<Route path="/department/:dept/:level" element={<ProtectedRoute><PostsPage
/></ProtectedRoute>} />
<Route path="/department/:dept/:level/post/:postId" element={<ProtectedRoute><PostDetail
/></ProtectedRoute>} />
```

4. Kotlin/Compose UI implementation

Android Kotlin Jetpack Compose:SignUpScreen.kt, CreatePostScreen.kt,

CreateSubspaceScreen.kt LiginScreen.kt(error when naming), SubspaceScreen.kt, PostsScreen.kt,

HomeScreen.kt, ProfileScreen.kt (all stored on compact disc)

5. WebSocket connection handling

```
@app.websocket("/ws")
async def websocket_endpoint(websocket: WebSocket):
await manager.connect(websocket)
try:
while True:
await websocket.receive_text()
pass
except WebSocketDisconnect:
manager.disconnect(websocket)
@app.get("/users", response_model= List[schema.UserOut])
def get_users(db: Session = Depends(get_db)):
return db.query(model.User).all()
class ConnectionManager:
def __init__(self):
self.active_connections: List[WebSocket]= []
async def connect(self, websocket: WebSocket):
await websocket.accept()
self.active_connections.append(websocket)
def disconnect(self, websocket):
self.active_connections.remove(websocket)
async def broadcast(self, message: dict):
for connection in self.active_connections:
await connection.send_json(message)
manager = ConnectionManager()
```

Appendix C: Installation and Deployment Guide

This appendix provides detailed instructions for setting up, installing, and running the **BenTalk** application from the files provided on the accompanying **Compact Disc (CD)**.

The CD contains the complete source code for the **Backend (FastAPI + SQLite)**, the **Web Frontend (React)**, and the **Android App (Kotlin / Jetpack Compose)**.

1. Contents of the Compact Disc

Folder Name	Description
BUILDFILES PSC2105330	**Contains all backend source files for the FastAPI server, database models, and configuration scripts.**
MyApplication	**Contains the Kotlin / Compose implementation of the BenTalk Android mobile app.**
PYTHON BACKEND CODE	**Contains all backend source files for the FastAPI server, database models, and configuration scripts.**
BENTALK web app version 2.1	**Contains all source files for the React web version of BenTalk.**

2. Setting Up the Development Environment

Before running the applications, ensure that the following software is installed on your system:

Required Software

- **Python 3.10+** (for backend)
- **Node.js 18+** and **npm** (for React web frontend)
- **Android Studio 2022.3+** (for Android app)
- **SQLite** (bundled with Python)
- **Git** *(optional, only if cloning or version control is needed)*

Steps

1. **Insert the Compact Disc** into your computer.. **Copy the entire project folder** (e.g., "BenTalk") to your computer's local drive, such as:
   C:\Users\<YourName>\Documents\BenTalk

2. Open separate terminals or IDEs for:

   - Backend (Python / FastAPI)
   - Web Frontend (React)
   - Android (Android Studio)

3. Backend Installation and Configuration (FastAPI + PostgreSQL)

Open a terminal and navigate to the backend folder:

cd BenTalk/backend

1. Create and activate a virtual environment: python -m venv venv venv\Scripts\Activate # On Windows # or

source venv/bin/activate # On macOS/Linux

2. Install dependencies: pip install -r requirements.txt *(If no requirements.txt file is present, install manually:)* pip install fastapi uvicorn sqlalchemy pydantic
3. Download PgMyAdmin:

Follow the installation process

Create a user

Create a database and set a password (you will automatically become the "super" user

Change the fill the name and password of your Postgres database into the database.py

4. Start the backend server: uvicorn main:app --reload
5. Open your browser and visit: http://127.0.0.1:8000/docs to view the interactive API documentation.

4. Web Frontend Installation and Deployment (React JS)

In a new terminal window, navigate to the frontend folder:

cd BenTalk/frontend

1. Install dependencies: npm install
2. Run the application in development mode: npm start. This launches the web version of BenTalk in your browser at: http://localhost:3000
3. To build a production version (optional): npm run build
4. The optimized build will appear inside the build/ folder, ready for deployment on web servers.

5. Android App (Compose UI) Setup and Execution

1. Launch **Android Studio**.and click **"Open an Existing Project"** and navigate to: BenTalk/android/
2. Allow Gradle to sync and download dependencies automatically.
3. Connect an Android device or start an emulator.
4. Click run to build and install the app.
5. The app will launch automatically on the selected device or emulator.

To export an APK: Build → Build Bundle(s) / APK(s) → Build APK(s)

The generated APK will be located under: app/build/outputs/apk/debug/

6. Deployment Notes

- The backend server (FastAPI) must be running before opening the web or Android versions to ensure data exchange.
- If running web and backend separately, ensure **CORS** is properly configured in the backend.