ILUVUS

Made by:

Team JIC 4321

Shreya Devaraju, Christeena Joby, Devika Papal, Roham Wahabzada, Sahil Virani

Client:

James Elliot

Repositories:

Server: https://github.com/ILUVUS/iluvus-backend-api
Application: https://github.com/ILUVUS/iluvus-react-native

Table Of Contents

Table of Figures

Terminology

The following dictionary is a collection of the terminology and corresponding definitions that this document uses.

A	API	Application Programming Interface; a connection between computers; our backend will be exposed to clients through an API.
	Android	A mobile phone operating system.
B	Backend	The web service responsible for connecting client devices, collecting data from clients, and distributing information to clients
D	Docker	An open platform for developing, shipping, and running applications.
F	Frontend	The user interface and presentation layer of a software application, visible to and interacted with by users.
G	Google Cloud Platform (GCP)	Google Cloud Run for deploying and running containerized applications.
	Google Bucket	A scalable and secure object storage solution provided by Google Cloud Platform.
	GitHub	GitHub's built-in CI/CD solution for automating workflows, including building, testing, and deploying applications.
H	HTTP	Hypertext Transfer Protocol. HTTP is a protocol for transmitting media, like HTML, between web browsers and servers.
I	iOS	Apple’s mobile phone operating system.
J	Java	An object-oriented programming language known for its platform independence and used for developing a wide range of applications.
	JavaScript	A high-level programming language that is often used in frontend development.
	JSON	JavaScript Object Notation is a lightweight data interchange format that is easy for humans to read and write, and easy for machines to parse and generate.
M	Mongo DB	A type of NoSQL database management system that stores data in JSON-like documents.

	MVC (Model-View-Controller)	A software design pattern commonly used for developing user interfaces that divides the related program logic into three interconnected elements.
N	NoSQL	Stands for “Not SQL.” A category of database management systems that are non-relational and follow a less rigid structure than SQL databases.
P	Postman	An application designed for testing and managing APIs, providing a user-friendly interface for making HTTP requests and analyzing responses.
R	React Native	A JavaScript framework for building mobile applications that can run on both iOS and Android platforms.
	RESTful API	An architectural style for an application program interface (API) that uses HTTP requests to access and use data.
S	Spring Boot	A Java-based framework for building robust and scalable backend applications.
T	Tailwind CSS	A utility-first CSS framework for rapidly building custom designs.
U	UI	User Interface refers to the visual elements and interactive components of a digital system or software application that enable user interaction.

Introduction

Background

In today's landscape, existing social media platforms continuously fail to provide a platform that highlights and connects people from underrepresented and minority groups. Last year, ILUVUS was designed to fill this gap and offer an application to uplift, celebrate, and recognize these underrepresented communities. As a continuing team, our team was impressed with the progress made by the previous group in laying out the foundation for the app. The goal for our portion of the project is to flesh out the existing application to both fully embody the mission of the application and be completely robust and ready in time for launch. By building on the app's foundation, our team aims to create a more engaging user experience that offers a welcoming space for all users. In doing so, we hope to create a safe and empowering platform targeted to everyone who needs this type of space. This semester, our team aims to add the following features:

1. Allow users to join new general and professional communities catered to their interests where they can discuss and share content in unique ways.
2. Add fact-checking features so that hate speech and disinformation are minimized on the application.
3. Expanding on current features of the app: Adding more user customization, user control over their settings, user posting & notifications, improving user reporting features, etc.

Our team's vision is to create a social media platform that promotes positivity for its users. We aim to not only add features that reduce harmful content on the application but also encourage users to be more open, accepting, and celebratory of each other. ILUVUS will be available both on IOS and Android to ensure that any user can join.

Document Summary

Within this document both the **static and dynamic architectures** of the ILUVUS app are explained. Our diagram for the system architecture shows how technologies such as React Native, Tailwind CSS, Spring Boot, MongoDB, Docker, and Google Cloud Platform (GCP) work together on both the frontend and backend of the app. Our diagram for the dynamic architecture shows a sample scenario of a user trying to create a new community. It diagrams what is shown to the user and what is happening on the backend. It also explains the different stages that go on in the creation of a community. The **Data Storage Design** will include diagrams that show how data is stored on our backend MongoDB database. This will include unstructured data such as posts, likes, comments, etc. The **Component Detailed Design** will provide more insight on how each component in our app works, and their associated attributes, methods, and relationships. The **UI Design** will focus on how our users see and interact with the different screens on our app.

System Architecture

Introduction

The main goal of the ILUVUS system architecture is to provide a scalable and smooth user experience to ensure that our platform's true mission can shine through and it can become a safe space for people of all backgrounds to foster a collaborative and supportive community. Our system takes advantage of the best features within the technical tools used to provide an intuitive user experience. The system has multiple layers, frontend, backend, and databases, that operate independently but communicate with each other in a fast and efficient manner to meet and exceed user needs. We utilize RESTful API's and dual-layered storage solutions to address any scalability concerts and adapt to a growing user base or future updates required.

In the following sections, we present two diagrams: the Static Diagram and Dynamic Diagram. These two diagrams demonstrate how different components interact with each other to accomplish application tasks and procedures.

Our system architecture is divided into three main layers: the Front-End, Back-End, and Data Storage. The Front end, using React Native, handles all UI interactions and performs requests to the backend for managing platform content. The backend processes these requests using Spring Boot and handles any business logic related to API's and Data Storage. The final layer, the Data storage part, handles all storage concerns using MongoDB for smaller usage needs and Google Cloud Bucket for large media files.

Rationale

The ILUVUS system architecture is made up of various static and dynamic components that make up the entire platform. The user interface, the front-end, is made using React Native and this allows cross-platform functionality development across IOS and Android devices. It has various functions and components that allow us to use reusable buttons, inputs, and screen designs.This sort of design improves code maintainability and ensures consistency in the UI across different screens and user roles. Along with React, we are also using the Tailwind CSS library for creative styling tools that are easier for responsive and visually appealing user interfaces. Tailwind's usability first t approach allows for rapid prototyping and easier applicability to accessibility standards such as color contrast and screen layout clarity. The front end also used local storage to store user authentication information such as logins, to reduce log-in times and better the user experience. While AsyncStorage is used mainly to cache tokens for faster logins, we do not store any sort of sensitive information like passwords or personal data on the device. These saved data are cleared on logout and are securely transmitted using HTTPS to protect against any outside attacks.

For the backend of our platform, we have utilized Spring Boot and deployed it in a container for centralized hosting. The backend uses the Model-View-Controller (MVC) architecture to enhance performance but it does not include the View component to account for speed and efficiency. By separating general logic from data access and request handling, MVC helps us isolate concerns and scale backend independently from other sections. It serves as the main communication for the UI and uses RESTful API for HTTP responses such as GET and POST. All the API requests go through the backend and it processes the data from the databases. Each API endpoint always performs input validation and user authentication before executing any database operations to help prevent unauthorized access. For any read or write information, the UI must provide accurate and specific information for the backend to process and approve those requests and modify the database.

The system has a dual-layer storage system with MongoDB and Google Cloud Bucket. MongoDB is the primary database to store user, community, post, interest, and other data. This secure storage allows the application to maintain data security and privacy standards as per industry conventions and our client's requirements. Furthermore, it is a flexible and a very scalable solution. However, for very large media, the backend will make calls to Google Cloud Bucket for storage. These calls also save the media URLs in MongoDB for reference.To give an overall picture of the system workflow: the frontend first communicated with the backend issuing HTTP requests to send and receive data. Then, the media files process and store these data in Google Cloud or MongoDB. All HTTP communication is conducted over HTTPS to ensure encryption while transmitting, and only authenticated sessions can trigger any sort of modifying operations. The backend also validates and processes these requests and uses controllers and the database to ensure optimal and efficient operation. Finally, the backend sends JSON format responses to the front to render a view to the users. This architecture is flexible, and efficient and provides a smooth user experience.

Static Architecture Diagram

We decided to use a static system diagram to showcase how the various components work together at a high level. This way, with an easy visual, a viewer can better understand the basic dependencies occurring – as the frontend, backend, and data storage work together to power the application – without getting too lost in the details.

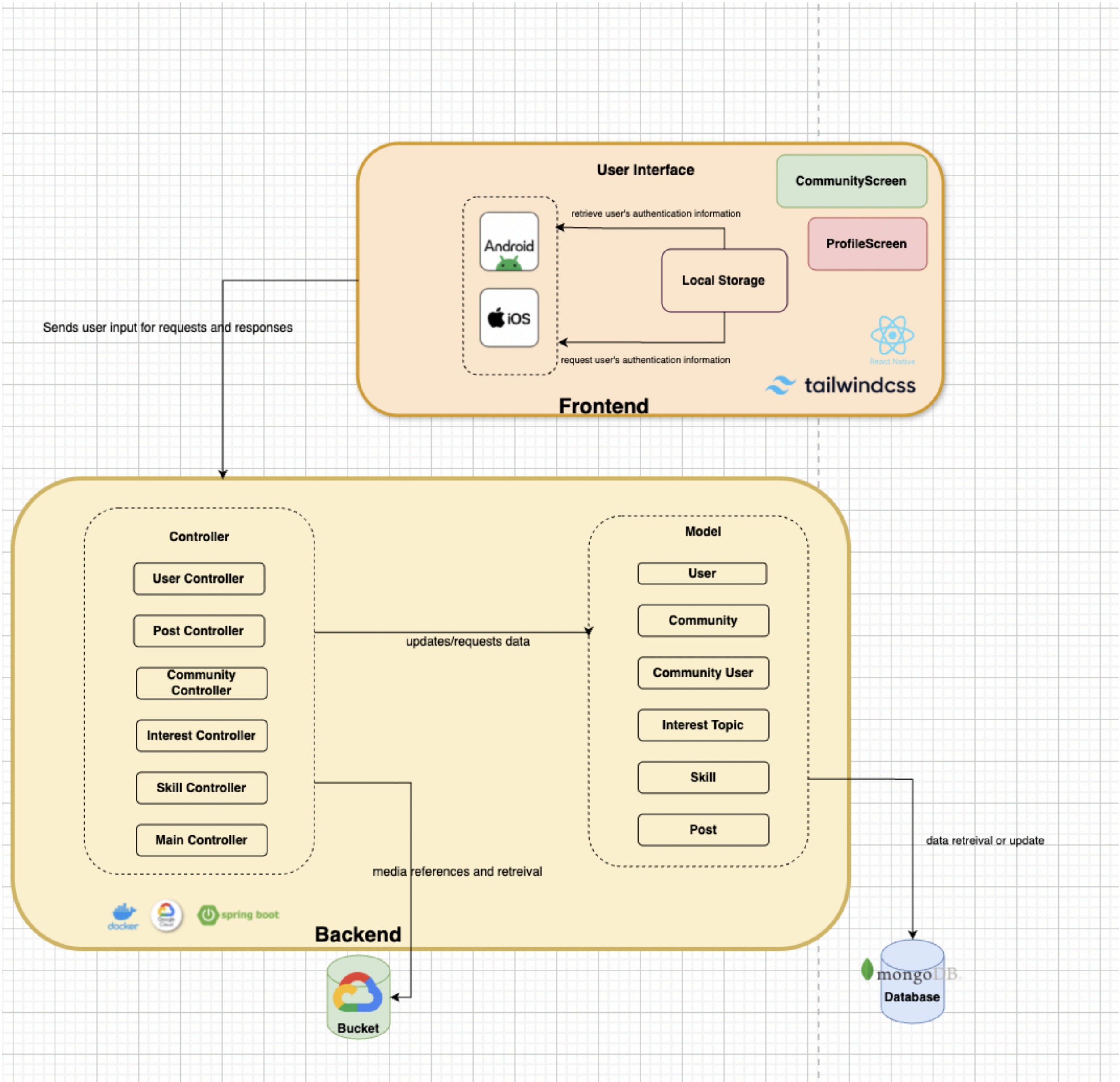


Figure 1: Static System Architecture Design for ILUVUS

Dynamic Architecture

The diagram below shows the flow for creating a new community in ILUVUS, which is a feature exclusively available to professional users (note that a professional user can be verified). Our team chose to use a system sequence diagram to show the correct relationships between the backend and the frontend, more specifically how the user interactions, UI layer, and the different controllers work together in succession to create a seamless experience for the end user.

The process of creating a new community begins once the user is logged into his/her ILUVUS account. From the home screen, the user must click on the community tab, which will redirect them to the CommunityScreen. From there, the user can select the appropriate button to create a new community (indicated by the "+" icon). Once this button is pressed, the user is prompted to fill out a form for the new community with relevant details. Once those details are filled out by the user and submitted, the *createCommunity()* function is invoked, which ultimately communicates with the backend API - sending the data over to a controller and eventually adding a new entry to the community database (MongoDB Atlas). Once a new community has been successfully added to the database, a success message is propagated back up to the UI layer.

Within the application, this is a feature exclusively available to professional users. First, we start by logging the user in. When a user enters their credentials, this information is sent to the controller which communicates with the database to validate the login information. It then returns a user Id. If the user id is non-empty, this means that the login was successful, and thus, the user can access their account. If not, they get a failure message. The logged-in user can then click on the community tab. When this community tab is accessed, the community screen communicates with the database via the controller to check if the user is professional (i.e. *isProfessionalUser()*). If this returns as true, the community screen provides the user with the option to create a community. If not, this option is absent.

The user initiates the create new community sequence by entering the details of the community and then hitting the Create new community button. This invokes the *createCommunity()* function that communicates with the interface layer and sends over the data to the controller, eventually adding a new entry to the community database. The database creates a unique ID for the community and propagates a success message back to the UI layer.

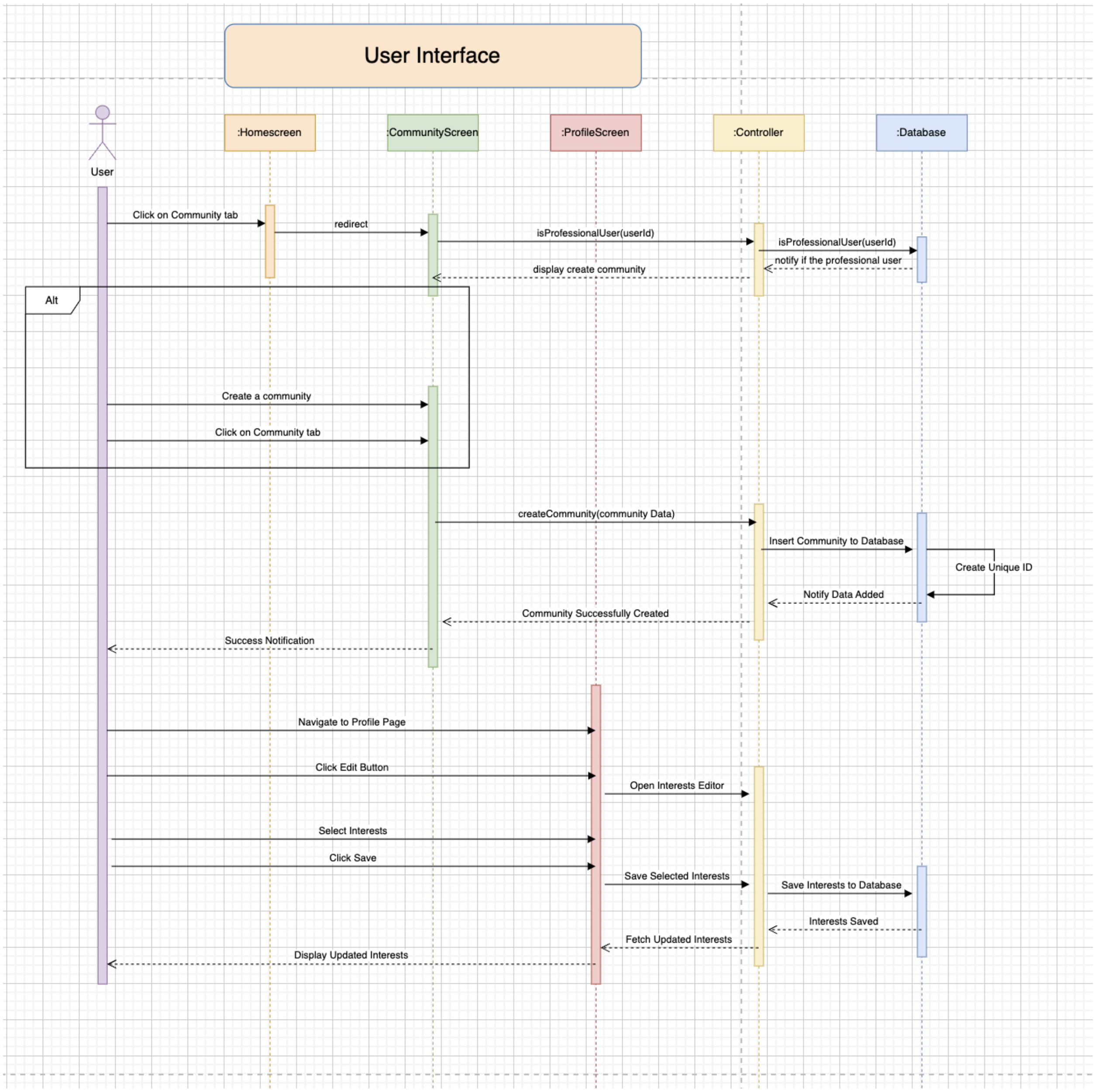


Figure 2: System Sequence Diagram for ILUVUS

Component Design

Introduction

In the last section, we covered the basic system-level architecture of the system. In this section, we dive deeper, presenting two component diagrams to showcase the relationships between the main components in ILUVUS. The first (Figure 3) is a class diagram outlining the different static components' design for the application - both frontend and backend - for the key features of the application. It will go into detail showing the different classes, attributes, and relationships in the entire system structure. The second (Figure 4) is a sequence diagram that highlights a typical user interaction with the interface where it concerns community and post-management. This new diagram goes deeper into the lower-level components and showcases new interactions being added in particular creating and displaying a post or community as well as reporting posts.

Static Component Diagram

For the static component design, our team decided to use a class diagram to show the static elements in the component design of our main key features in alignment with the MVC system architecture that we detailed earlier. This way, viewers can understand the complexities of the overall system design at a deeper level as it pertains to important association and inheritance implications for the overall application function.

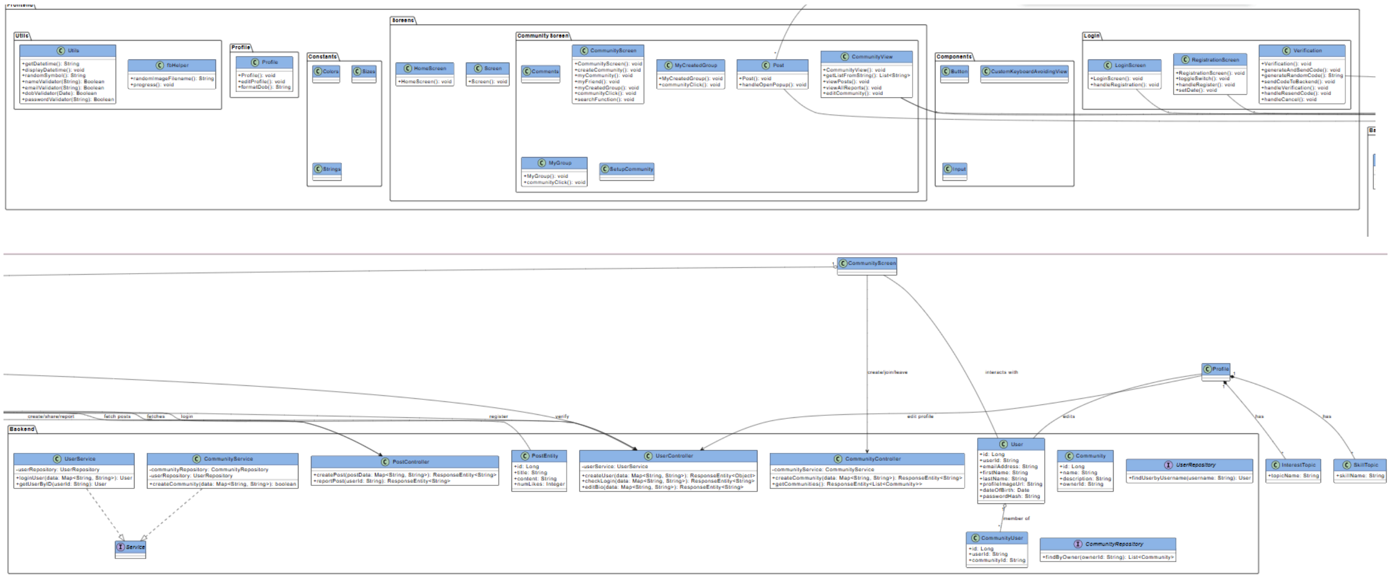


Figure 3: Static Component Diagram using Class Diagram (Full diagram viewable here: LINK)

In the diagram above, our static components are divided into two distinct components: frontend and backend, and are differentiated accordingly (Figure 3). All the controllers shown in the diagram act as tools for frontend requests to access and change the database. Each of these controllers has an associated service class that not only has an interface object to access the database but also handles the logic to update and validate models, essentially acting as a mediator between everything above and below the class. For instance, the UserService class uses User Repository and User Object to handle requests from the User Controller and so on for the Post Controller and Community Controller. To save space, not all controllers have been shown but the same logic exists for the other controllers and services (Interest, Skills, and Main Controllers). Each of these relationships has been denoted by lines associated with the classes.

To ensure flexibility and efficiency, the frontend must call Restful API to contact the Backend and access the database. Each of the backend controllers contains the associated API and calls their respective Service classes to accomplish the business logic of that API. The API then returns the object in JSON format to the front end so that the front end can properly access and display the data to a user.

Dynamic Component Diagram

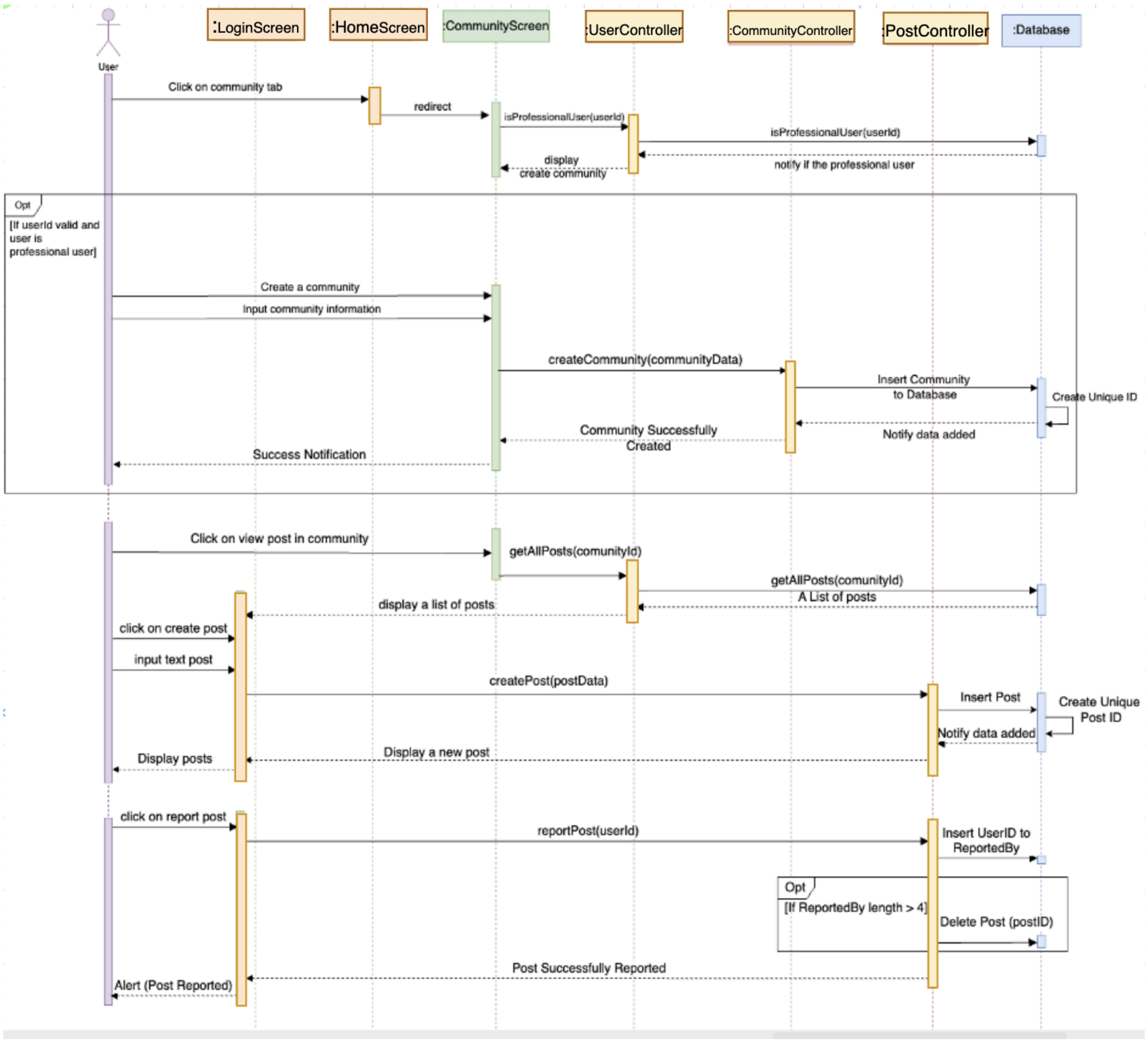


Figure 4: Sequence Diagram as Dynamic Diagram

The sequence diagram above demonstrates how a user interacts with the application in order to manage communities and posts. We chose to use a Sequence Diagram to clearly show the necessary actions between the user, the controller, and the database. Ultimately, this helps provide a visualization of how the different components work together and communicate with each other in the system. The user starts by selecting the community tab while on the post screen which then leads them to the community interface. At this stage, the system checks if the user is a professional user using the isProfessionalUser(userId) function. If the user is a professional then the option to move forward and create a new community is displayed. After the user inputs the community details the **createCommunity(communityData)** function processes the information and the controller communicates with the database to store the new community. A unique communityId is also created during this step. Once the community is created a notification alerts the user of the successful community creation.

After the community is created, a user can view the posts related to that community. By attempting to view the posts, the controller is signaled and fetches the different posts using the getAllPosts(communityId) function. The database pulls the posts, and they are then displayed on the community screen. The user can scroll to view all the posts.

If the user decides to create a post then would input the contents of their posts on the screen, which would then trigger the **createPost(postData)** function upon submission. The background process is similar to the community creation, where this time the controller inserts the post in the database, assigning it a unique ID, and then displays a confirmation message to indicate a successful creation. The sequence diagram uses solid black lines to indicate the processes going on at the same time.

The last part of the diagram shows how a user is able to report a post. When a post is reported, the postController checks if the post has been reported more than five times, indicating that it has been reported by multiple users. If this condition is met the post is deleted from the post database. Otherwise, the counter is updated and the username of the user who reported it is stored in a list. A response is then sent back to the user indicating that their report has been received.

Data Design

Introduction

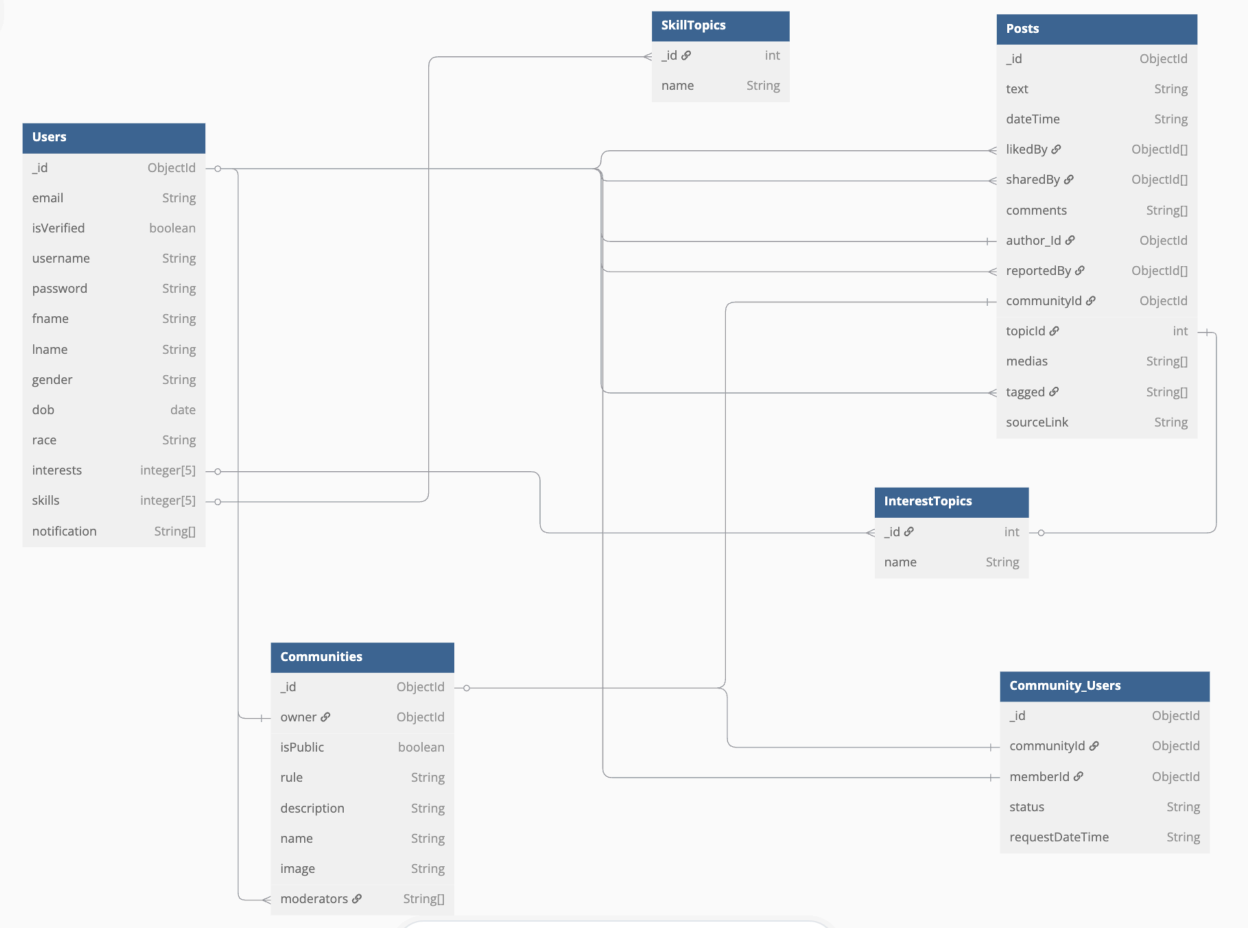
The Data Design section is divided into four key sections: Database Use, File Use, Data Exchange, and Security. This section explains how ILUVUS manages data storage efficiently while ensuring scalability, performance, and security.

For ILUVUS, we have chosen MongoDB as our primary database over a traditional SQL-based system for the following reasons:

- **Flexible Schema and Data Structures:** MongoDB allows us to store data in a JSON-like format, making it highly adaptable for a social media platform where user profiles, posts, and interactions may evolve over time.
- **Scalability for Large Data Volumes:** NoSQL databases like MongoDB are optimized for handling high-volume workloads, which is crucial for ILUVUS as it grows to support millions of users and user-generated content.
- **Horizontal Scaling:** Unlike traditional SQL databases that rely on vertical scaling, MongoDB supports sharding, which enables data distribution across multiple servers to handle increased traffic and data growth efficiently.
- **Write Availability and Eventual Consistency:** MongoDB offers replication and automatic failover, ensuring that ILUVUS remains accessible even during heavy traffic loads. It also provides eventual consistency, which prioritizes availability while ensuring data integrity.
- **Developer-Friendly Integration:** As a NoSQL database, MongoDB's JSON-like document model aligns well with JavaScript-based technologies like React Native and Node.js, making it easier for our team to develop, maintain, and iterate on features quickly.

In addition to MongoDB, ILUVUS leverages Google Cloud Bucket for storing large media files, ensuring seamless file management and retrieval. The following subsections provide a detailed breakdown of how data is stored, exchanged, and secured within the ILUVUS ecosystem.

Database Use



SkillTopics	
_id	int
name	String

Posts	
_id	ObjectId
text	String
dateTime	String
likedBy	ObjectId[]
sharedBy	ObjectId[]
comments	String[]
author_Id	ObjectId
reportedBy	ObjectId[]
communityId	ObjectId
topicId	int
medias	String[]
tagged	String[]
sourceLink	String

Users	
_id	ObjectId
email	String
isVerified	boolean
username	String
password	String
fname	String
lname	String
gender	String
dob	date
race	String
interests	integer[5]
skills	integer[5]
notification	String[]

InterestTopics	
_id	int
name	String

Communities	
_id	ObjectId
owner	ObjectId
isPublic	boolean
rule	String
description	String
name	String
image	String
moderators	String[]

Community_Users	
_id	ObjectId
communityId	ObjectId
memberId	ObjectId
status	String
requestDateTime	String

Figure 5: Document-Database Diagram for MongoDB Database

Each table in the diagram (Figure 5) above represents a collection of data. The lines denote reference connections between attributes. The three-pronged endpoint of a connecting line indicates the “many” side of a one-to-many relationship. All other connections represent a one-to-one relationship. For instance, many users can be tagged in a post but only one user can be the author for a post.

File Use

For this version of the application, images are being stored within the Google Cloud Bucket and the app itself will support all image types imported from the local device’s photo gallery. Ultimately, however, when uploading to the cloud, we use an algorithm to compress all images to a JPEG type. For the backend, our team is using Java so most of the files have a .java extension whereas the frontend uses React Native so almost all of those files have a .js extension.

Data Exchange

The data for this project is stored in a backend remote MongoDB database. The backend server communicates with the MongoDB database based on the CRUD model. Users can Create, Read, Update, and Delete data. The frontend communicates with the backend using HTTP requests, mainly through POST and GET requests. After a user is validated the backend server can pull data from the database and send it in JSON format to the frontend, or it can make changes in the database if necessary.

Security:

Currently, the app ensures that all personal details such as passwords are hashed so that even someone with access to the source code would not be able to view the passwords of users. HTTPS should be used over HTTP so that information cannot be intercepted when transmitted. It is important to make sure the channel is secure before sending any data. User login is required for this app. Upon sign in, a user's account is verified using a UserID check which ensures that only authenticated users can use the app.

Upon sign up, a user must create a username and password (hashed) and then verify their account with a code that has been sent to their email (Two-Factor Authentication). Currently, the only Personally Identifiable Information (PII) that is being protected is the password.

Given that this app is a social media site, the whole purpose is for users to share more about themselves with others. Ultimately, users can choose how much information they can share along with whom they share their own information (this includes date of birth, names, etc.). However, we understand that because ILUVUS's API handles sensitive personal data, implementing security measures such as API gateway security or authorization will be crucial to prevent misuse of data. Currently, this goes beyond the scope of our project due to time constraints, however, we suggest that our client place API request validation for API gateway and OAuth2.0 as a priority to implement in the period directly before and after the launch of the completed application to provide strong protection for users' data as the user base of the application continues to grow.

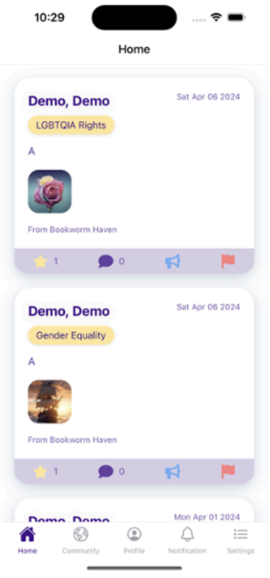
UI Design

Introduction

In this section, our team will present the User Interface (UI) of the ILUVUS mobile application, delving into the major screens that the user will interact with. Specifically, we focus on the main screen and the key features being released, such as community creation, post creation, and post interactions (such as reporting). Ultimately, by visually walking through the major components of the final application, this section should give the reader a good understanding of the team's approach and process when designing the larger interface, as well as a key preview of the user experience.

Screenshots and Descriptions + Heuristic Analysis

Main Screen - Screenshots and Description



Demo, Demo — Sat Apr 06 2024
LGBTQIA Rights
A
From Bookworm Haven
1 · 0

Demo, Demo — Sat Apr 06 2024
Gender Equality
A
From Bookworm Haven
1 · 0

Demo, Demo — Mon Apr 01 2024

Figure 6 Home Page with Interested-related Posts

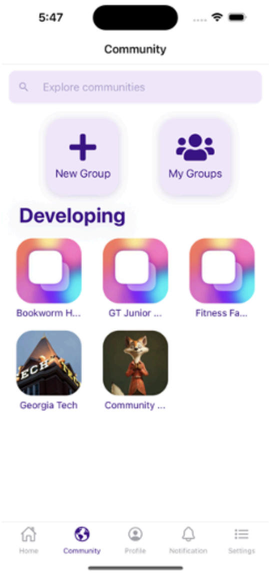
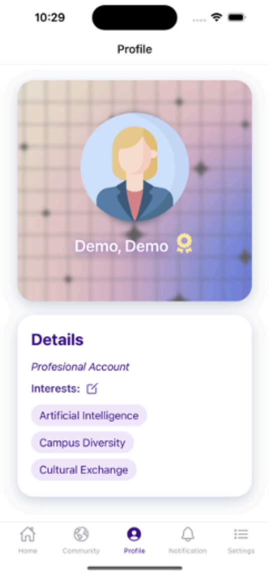


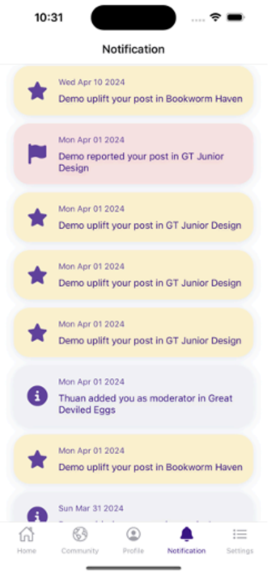
Figure 7 All Available Communities on ILUVUS



Demo, Demo

Details
Profesional Account
Interests:
Artificial Intelligence
Campus Diversity
Cultural Exchange

Figure 8 User Profile with Selected Interests



Wed Apr 10 2024
Demo uplift your post in Bookworm Haven

Mon Apr 01 2024
Demo reported your post in GT Junior Design

Mon Apr 01 2024
Demo uplift your post in GT Junior Design

Mon Apr 01 2024
Demo uplift your post in GT Junior Design

Mon Apr 01 2024
Demo uplift your post in GT Junior Design

Mon Apr 01 2024
Thuan added you as moderator in Great Deviled Eggs

Mon Apr 01 2024
Demo uplift your post in Bookworm Haven

Sun Mar 31 2024

Figure 9 User's Notifications

Once the user has logged in, they can go to their profile page(Figure 8), where they will be able to see a pen and paper icon. This is the interests tab. Here, the user can select their interests of choice from different topics. Once they are done, they can see that their home page(Figure 6) will be filled with posts that reflect their selected interest topics. In the community page(Figure 7), users can explore and participate in various interest-based communities within the social media platform. The notification screen(Figure 9) serves as a real-time update center, informing users about interactions and activities related to their accounts.

Main Screen - Heuristic Analysis

Figure 6: Home Page with Interest-related Posts

1. Visibility of System Status - Each post displays a timestamp (e.g., "Sat Apr 06 2024") and interaction metrics such as likes, comments, and report counts. This keeps the user informed about how recent the posts are and how others have interacted with them. It helps users quickly assess what content is new or popular.
2. Match Between the System and the Real World - Topics like "LGBTQIA Rights" and "Gender Equality" are labeled with real-world terminology that users immediately understand. This alignment with everyday language makes the interface more approach
3. Aesthetic and Minimalist Design - Only essential elements are shown: post content, topic tag, media, and engagement icons. The spacing and layout reduce cognitive overload and focus the user’s attention on relevant content.

Figure 7: All Available Communities on ILUVUS

1. Recognition Rather Than Recall - Communities are represented with visual icons and names, allowing users to recognize them without having to remember or type them. The layout supports browsing and quick selection.
2. Consistency and Standards - The UI follows platform standards: a bottom navigation bar, clear icons for community actions, and intuitive organization. This consistency ensures that returning users or those familiar with other apps can navigate effortlessly.

3. Flexibility and Efficiency of Use - Professional users have access to a "New Group" button, while regular users only see options relevant to them. This role-based customization makes the interface adaptable to varying needs.

Figure 8: User Profile with Selected Interests

1. Match Between the System and the Real World - The word "Interests" is easy to understand, and the tags for selected interests (e.g., "Artificial Intelligence", "Campus Diversity") use everyday concepts instead of internal jargon.
2. User Control and Freedom - The edit icon next to "Interests" gives users the ability to modify their selected topics at any time, making the system feel responsive and under their control.
3. Aesthetic and Minimalist Design - The profile is not cluttered. Only key details—name, type of account, and selected interests—are displayed in a clean card format.

Figure 9: User Notifications

1. Visibility of System Status - Each notification is time-stamped and shows clear updates on user interactions (e.g., "Demo uplifted your post"). This lets users stay up to date with minimal effort.
2. Recognition Rather Than Recall - Users don't need to remember what happened previously—the system clearly shows updates with user names, community names, and action icons.
3. Consistency and Standards - Notifications are visually consistent, each using the same card style, icon placement, and action wording, which supports predictability and trust.

Creating a Community - Screenshots and Description

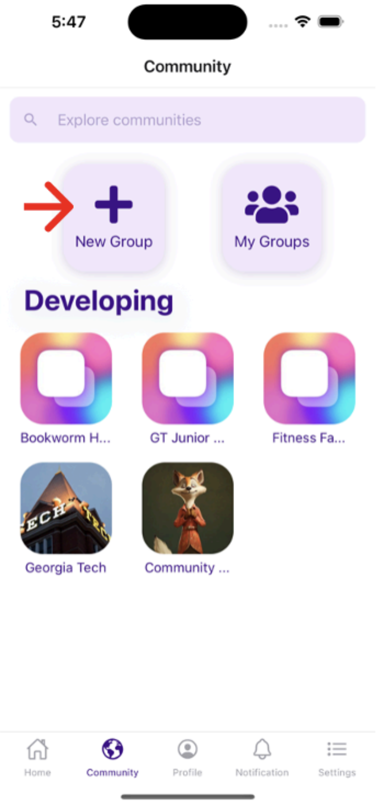


Figure 10 Professional User Can Create a New Community

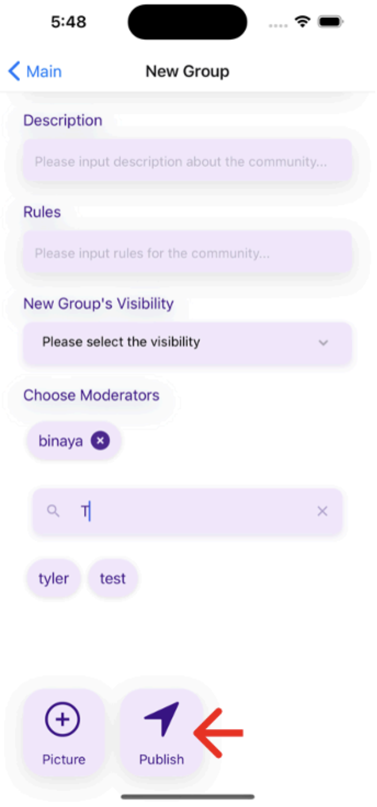
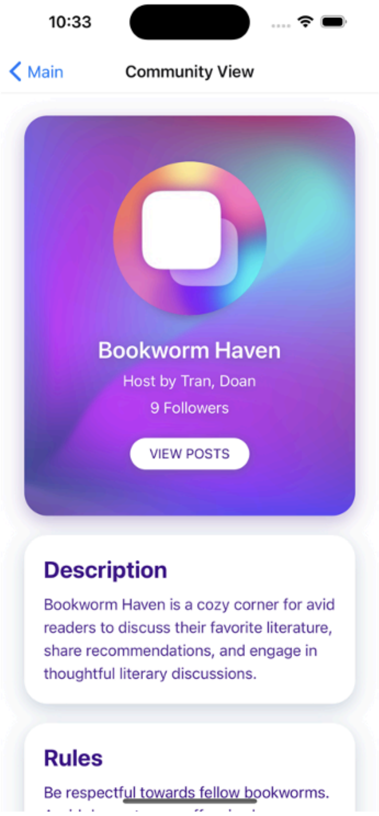

5:48
Main
New Group
Description
Please input description about the community...
Rules
Please input rules for the community...
New Group's Visibility
Please select the visibility
Choose Moderators
binaya
T
tyler
test
Picture
Publish

Figure 11 Professional Can Choose Moderators and Push the Community



Description

Bookworm Haven is a cozy corner for avid readers to discuss their favorite literature, share recommendations, and engage in thoughtful literary discussions.

Rules

Be respectful towards fellow bookworms.

Figure 12 Community View After with its Information

Another tab that the users can access once they login or create an account is the community tab(Figure 10). This community tab is different for professional users (those who have logged in with a professional email id) and regular users. The regular users may see a search tab, as well as the list of groups they have joined. However, a professional user will see a button that gives them the option to create a new group. When the user clicks this button, they will have to fill in four required inputs for a new community which are name, description, rules, and visibility(Figure 11). The four regular text inputs will show an error dialog if they are left empty. One of the four required inputs allows the user to decide the New Group's Visibility which can be either Public or Private. Public visibility will allow everyone to join the community without the need for prior verification from the group owner. The content will also be publicly visible. Private visibility groups will need the new users to request the owner or the moderators to join the community. There are also two optional inputs -- moderators and profile pictures. We give options for having moderators as it is hard to regulate a large community without human moderation. To add a moderator to the community, the professional user can search for users by their username from all the users on ILUVUS. Selected moderators will appear with an “X” icon on the right of their username and if user want to unselect the moderators, they can click “X” for cancel. The last step is to add the community profile picture by clicking on the Picture button on the bottom left. A native image picker will allow the creator to select their preferred image from their image library. Finally, they can use the Publish button to create the community, which will take them to the newly created community screen (Figure 12)

Creating a Community - Heuristic Analysis

Figure 10: Professional User Can Create a New Community

1. Consistency and Standards - The "+" icon is a universal standard for adding new items. Its placement in the top-left and labeling as "New Group" aligns with common user expectations.
2. Match Between the System and the Real World - The use of terms like "Group", "Community", and "New Group" make sense in everyday language. There is no technical jargon to confuse the user.
3. Flexibility and Efficiency of Use - This screen is adapted for professional users. It offers them specific tools (like group creation) that regular users don’t see, making the experience efficient and role-specific.

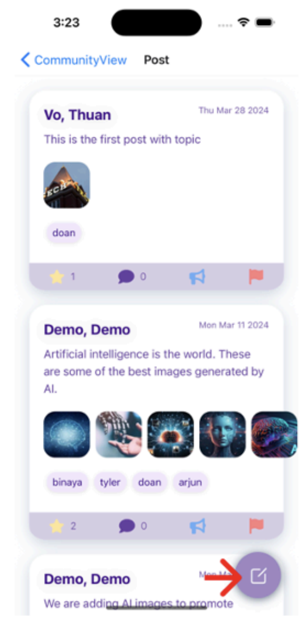
Figure 11: Choose Moderators and Push the Community

1. Error Prevention - Required fields (e.g., Description, Rules) prevent submission if left empty, helping users avoid errors during group creation.
2. Visibility of System Status - As users select moderators, they instantly appear with an "X" icon, showing which moderators have been chosen. This provides immediate feedback.
3. User Control and Freedom - Users can remove moderators before publishing by tapping the "X". This gives them full control over final submission.

Figure 12: Community View with its Information

1. Recognition Rather Than Recall - All community details are displayed upfront—users don't need to recall what they input previously. This is helpful for transparency and confirmation.
2. Aesthetic and Minimalist Design - The view displays only key sections—Description, Rules, Host—which reduces cognitive load and makes the interface easy to scan.
3. Visibility of System Status - The "VIEW POSTS" button clearly shows users the next action they can take after reading about the community.

Creating New Posts - Screenshots and Description



Vo, Thuan — Thu Mar 28 2024
This is the first post with topic
doan

Demo, Demo — Mon Mar 11 2024
Artificial intelligence is the world. These are some of the best images generated by AI.
binaya tyler doan arjun

Demo, Demo
We are adding AI images to promote

Figure 13 Community Members Can Create Posts

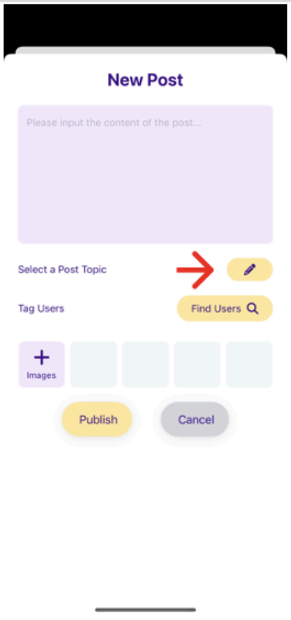


Figure 14 Select a Topic for Post

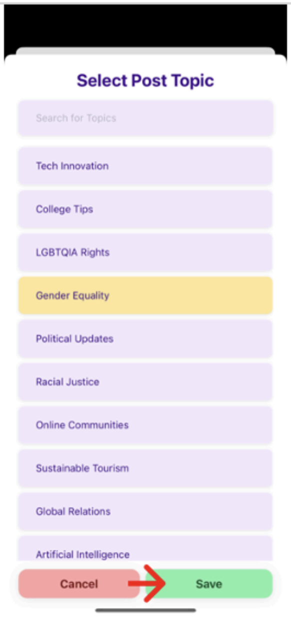


Figure 15 Select and Save One Topic per Post

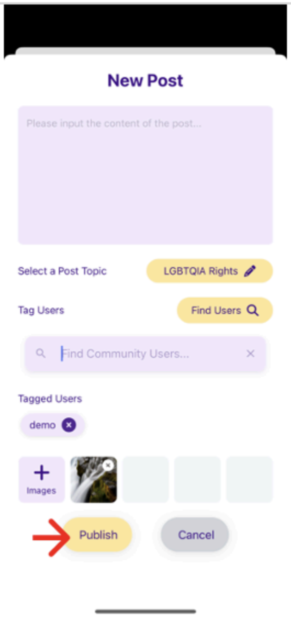


Figure 16 User Can Tag Other Users and Add Medias

Once you join the community as per Figure 10, 11, and 12, you can view all the posts within it. On the Post View of a joined community, there is a Paper and Pencil icon on the bottom right as seen in Figure 13. Clicking on this button will show a new post creation form. A new post requires 2 inputs such as content and topic. The user can input the content in the big purple rectangle box. To select a topic, they can click on the purple pen icon which will navigate to a searchable list of topics. After selecting a topic, the user can either save the new topic or cancel the previously selected topic. In the creating post form, the user can search for users within the community to tag by using the search bar under the topic button. Next, the user can select up to five images along with post content. Lastly, users press on “Publish” to create and publish the post to the community or cancel to discard input content.

Creating New Posts - Heuristic Analysis

Figure 13: Community Members Can Create Posts

1. Visibility of System Status - The pencil icon is always available and clearly visible in the bottom-right corner, indicating the action to create a new post.
2. Recognition Rather Than Recall - Icons for likes, comments, and reports are easily recognizable and align with what users expect from social platforms.
3. Aesthetic and Minimalist Design - Posts are presented with spacing, media, and content clearly separated, creating a clean reading and browsing experience.

Figure 16: Tag Users and Add Media

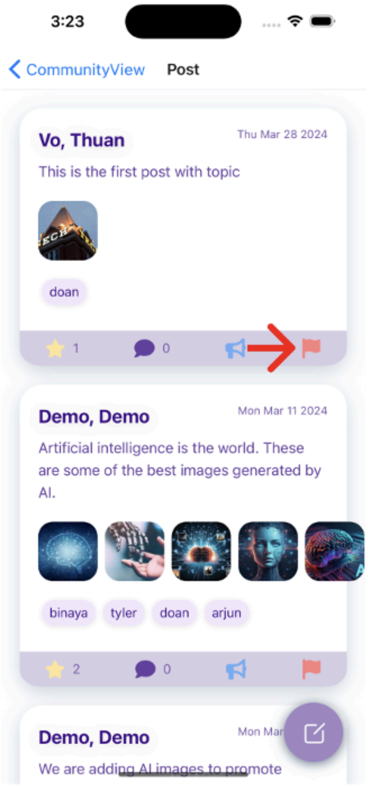
1. Flexibility and Efficiency of Use - Users can choose to tag others, add images, or just write content—allowing for personalized, flexible post creation.
2. Recognition Rather Than Recall - Previously tagged users and selected media are displayed immediately, so users don’t have to remember who they added.
3. Visibility of System Status - The interface updates in real-time as images are selected or tags are added, letting users know their changes have been saved.

Figures 14–15: Select and Save Post Topic

1. Error Prevention - Users are restricted from selecting more than one topic, which avoids confusion and maintains topic clarity.
2. User Control and Freedom - The "Cancel" button allows users to go back without saving a topic, enabling them to undo actions freely.

3. Consistency and Standards - The interface uses familiar list selection behavior, allowing users to select and confirm their choice with a standard Save button.

Reporting Violated Content - Screenshots and Description


3:23

CommunityView Post

Vo, Thuan Thu Mar 28 2024

This is the first post with topic

doan

1 0

Demo, Demo Mon Mar 11 2024

Artificial intelligence is the world. These are some of the best images generated by AI.

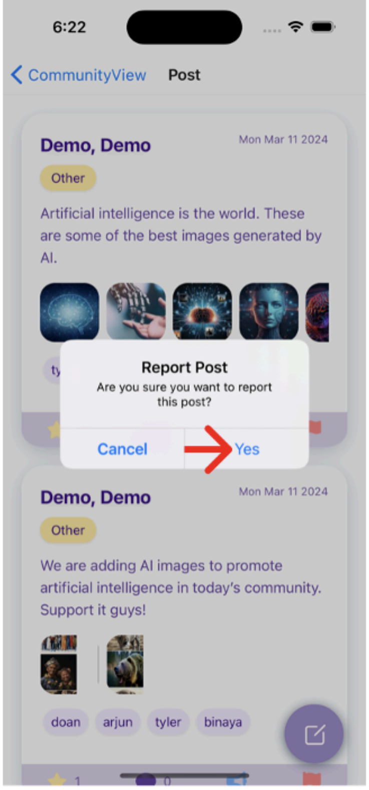
binaya tyler doan arjun

2 0

Demo, Demo Mon Mar

We are adding AI images to promote

Figure 17 Community User Can Report A Violated Post


6:22

CommunityView Post

Demo, Demo Mon Mar 11 2024

Other

Artificial intelligence is the world. These are some of the best images generated by AI.

Report Post

Are you sure you want to report this post?

Cancel Yes

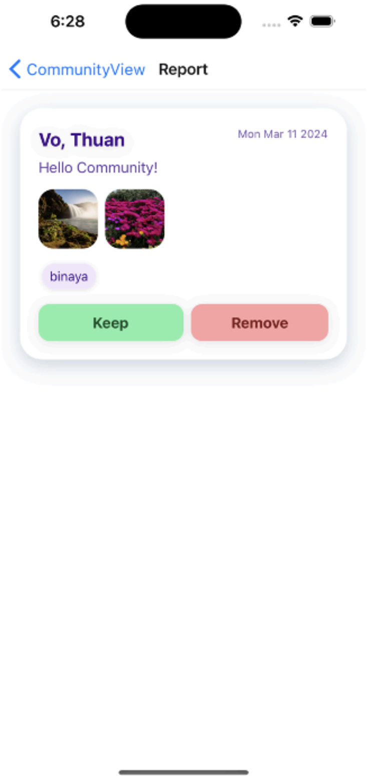
Demo, Demo Mon Mar 11 2024

Other

We are adding AI images to promote artificial intelligence in today's community. Support it guys!

doan arjun tyler binaya

Figure 18 Report Confirmation


6:28

CommunityView Report

Vo, Thuan Mon Mar 11 2024

Hello Community!

binaya

Keep Remove

Figure 19 Moderators View where All Reported Posts for Reviewing

In any community's Post View, a user can report inappropriate content by clicking on the red flag icon at the bottom right of a post. A confirmation message will pop up to ask for confirmation regarding the User's choice.

The owner or a moderator of the community can see the "Review Reports" option on their Community View, a screen that has a list of reports that exceed the flagging threshold of 5. They can then decide to keep the post if they believe the post content does not violate the community by clicking on the green “Keep” button. If they feel the post violates the rules of the community, they can click on the “Remove” button to delete the post permanently from the community.

Reporting Violated Content - Heuristic Analysis

Figure 17: Report a Violated Post

1. Consistency and Standards - The red flag icon for reporting is consistent with other apps and represents the reporting action universally.
2. Recognition Rather Than Recall - Users recognize the red flag icon immediately, without needing a tooltip or label.
3. Visibility of System Status - The icon becomes active and leads to a confirmation popup, giving immediate feedback on the user's action.

Figure 18: Report Confirmation

1. Recognition Rather Than Recall - A confirmation dialog prevents users from accidentally reporting a post. This is a critical safeguard.
2. User Control and Freedom - The "Cancel" option lets the user back out of the reporting process, avoiding unwanted actions.
3. Help Users Recognize, Diagnose, and Recover from Errors - The plain language confirmation message makes it clear what the user is about to do, ensuring full understanding before action is taken.

Figure 19: Moderator Review of Reports

1. Visibility of System Status - Moderators see reported posts in an organized view, with visual indicators and clear actions ("Keep", "Remove") available.
2. Help Users Recognize, Diagnose, and Recover from Errors - Moderators are able to reverse decisions by choosing "Keep" or finalize them by choosing "Remove", reducing the chance of wrongful deletion.

3. Match Between the System and the Real World - The task flow matches real-world moderation logic: review -> decision -> action. Labels like "Keep" and "Remove" are clear and intuitive.

Design Considerations

We designed ILUVUS with a focus on mobile use, easy-to-use navigation, and a structure that can scale and grow as our user base grows. The user interface was created using React Native and TailwindCSS to make it minimalistically aesthetic and responsive on mobile devices. Below, we highlight a few key design decisions and principles that guided our UI development:

Consistency:
We used well-known design patterns for user interfaces such as:

- Bottom-right floating action button for things like making posts or communities.
- Common icons like a pencil for editing, a flag for reporting, a bell for notifications, etc.
- We also made sure it's easy to go back to previous pages in the profile, search, and community sections with a common back button in the same corner in all these pages.

These options make it easy for users to feel at ease using the app without having to memorize the layout or lose their way navigating within the app since the UI is similar to what they are previously accustomed to.

Simplicity:
ILUVUS focuses on a simple user interface using soft cards and shadows that are used to organize information and show previews of posts and tags for interests. We use a clear and neat design with enough space and easy-to-read letters for a minimalistic view. Furthermore, we use different hues of our base colors to grab people's attention, like red for important alerts and purple for buttons you can click and navigate with. This makes sure the interface is not too complicated and helps users concentrate on the main information provided and the tasks within.

Error Prevention:
We set up immediate feedback throughout our app using pop-up messages for any user actions, such as sending a post or showing an error when fields are empty. We also use inline error validation to show messages right away if important information is missing. The app also sends messages that ask you to confirm before acting on something important, such as reporting a post

or leaving a group. These steps help avoid errors and lead users to have smooth experiences within the app.

Accessibility:

The buttons and items on the screen are designed to be easy to use on a mobile device, with the right sizes and space between them for tapping. Color contrast and text size are made to be easy to read in different types of light. Icons and interactions are designed to be user-friendly for everyone.

Scalability:

The app is built up in a way that makes it easy to scale and add new features. We use a plethora of reusable parts such as buttons, postcards, pop-up windows, etc in React Native and implement a flexible backend using Spring Boot, MongoDB, and Firebase that can easily add new features to better and improve the app as needs arise.

Appendix A

Team Information

Shreya Devaraju: Project Lead & Developer
Contact: sdevaraju6@gatech.edu

Shreya Devaraju is a third-year computer science major concentrating in Artificial Intelligence and Information Internetworks. As the current project leader for ILUVUS, Shreya has been heading up client communications as well as the design and development of the project for the past year. During the first semester of this project, she directed the vision refinement, user research, product mapping, and prototyping efforts. This semester, she's been planning the team's sprints and application architecture to align the app with the client's vision before launch. As a developer, she's utilized her backend expertise to flesh out and optimize the functionalities for key features such as user searching, profile maintenance, and direct messaging.

Christeena Joby: Designer & Developer
Contact: cjoby3@gatech.edu

Christeena Joby is a fourth-year computer science major concentrating in People and Media. As a designer & developer for ILUVUS, Christeena has been utilizing her expertise in design and frontend development across both semesters to brainstorm ideas, ensure the app adheres to user design principles, and support minimalistic and easy-to-use functionality that appeals to all types of users. This semester, she's been quickly identifying and resolving key issues in the frontend while also adding new user-focused designs for the additional features being added to the application. Some key features that she's worked on include direct messaging, user searching, and sharing posts.

Devika Papal: Designer & Developer
Contact: dpapal3@gatech.edu

Devika Papal is a fourth-year computer science major concentrating in Information Internetworks and Media. As a designer & developer for ILUVUS, Devika has been utilizing her expertise to streamline prototyping and ensure efficient devlopment in both backend and frontend development that is consistent across all portions of the application and adheres to industry standards. This semester, she's worked to identify and resolve issues on the frontend, such as loading buttons, and bugs on backend, such as loading time. Some key features that Devika has worked on include adding skills, topics of the day, filtering, and sharing posts.

Sahil Virani: Designer & Developer
Contact: svirani36@gatech.edu

Sahil Virani is a third-year computer science major concentrating in Artificial Intelligence and Information Internetworks. As a designer & developer for ILUVUS, Sahil has been utilizing his technical expertise to add insights and ensure correct and optimized functionality across all portions of the application. This semester, Sahil has not only identified and contributed to resolving key bugs, but he's also worked on key features such as source-checking, direct messaging, post searching, and user reporting.

Roham Wahabzadha: Designer & Developer
Contact: rwahabzada3@gatech.edu

Roham Wahabzadha is a fourth-year computer science major concentrating in System Architecture and Artificial Intelligence. As a designer and developer for ILUVUS, Roham has been utilizing his broad technical expertise to act as a technical advisor and optimize development across the platform. For instance, he's worked on resolving loading time throughout the semester and added key insights to important portions of development. Some key features that Roham has worked on include profile maintenance, source-checking, popular topics, and user warnings.

Appendix B

Response Codes

Code	Text	Description
200	OK	The request succeeded.
400	Bad Request	The server cannot or will not process the request due to something perceived as a client error.
403	Forbidden	The client does not have access rights to the content; that is, it is unauthorized, so the server is refusing to give the requested resource.
404	Not Found	The server cannot find the requested resource.
500	Internal Server Error	The server has encountered a situation it does not know how to handle.
503	Service Unavailable	The server is not ready to handle the request. Common causes are a server that is down for maintenance or that is overloaded.

POST user/create

Create a new user in the database.

Resource URL

https://testing-coh77z5j7a-ue.a.run.app/user/create

Request Information

Request Format: JSON

Resource Information

Response Format: String

Requires Authentication: No

Rate Limited: No

Parameters

Parameter	Description
username	Username of the user
fname	First Name of the user
lname	Last Name of the user
gender	Gender of the User
DOB	Date of Birth of the user
race	Race of the user
email	Email of the user
password	Password of the user

Example Requests

https://testing-coh77z5j7a-ue.a.run.app/user/create

Json:

```
{
        "dob": "2001-01-02",
        "education": "",
        "email": "bin@gmail.com",
        "fname": "Binaya",
        "friends": "",
        "gender": "Male",
        "groups": "",
        "hobbies": "",
        "interests": "",
        "lname": "Timsina",
        "password": "bin00000",
        "posts": "",
        "proEmail": "",
        "race": "Asian",
        "skills": "",
        "username": "bin",
        "work": ""
}
```

Example Response

“User created successfully”

“Please check the following fields: username,…”

POST user/verify

Verify the professional user or regular user.

Resource URL

https://testing-coh77z5j7a-ue.a.run.app/user/verify

Request Information

Request Format: JSON

Resource Information

Response Format: String

Requires Authentication: No

Rate Limited: No

Parameters

Parameter	Description
userId	ID of the user

Example Requests

https://testing-coh77z5j7a-ue.a.run.app/user/verify

Json:

```
{
        "userId": "65b7fed89cb7885873ade787"
}
```

Example Response

“Verified”, “Not Verified”

GET user/search

Get all the pending user information from a particular community.

Resource URL

https://testing-coh77z5j7a-ue.a.run.app/user/getMatchedUser

Resource Information

Response Format: JSON

Requires Authentication: No

Rate Limited: No

Parameters

Parameter	Description
filter	Data for search

Example Requests

https://testing-coh77z5j7a-ue.a.run.app/community/getMatchedUsers?filter=65d40edbab9c837874869dc4

Example Response

```
[{
        "fname": "Shreya",
        "lname": "Devaraju",
        "isVerified": true,
        "dob": "2024-02-20T05:00:00.000+00:00",
        "requestDateTime": null,
        "id": "65d40e26ab9c837874869dc1",
        "communityId": "65ef1eb0dff9da4a5baee4ca",
        "email": "shreyadevaraju@email.com",
        "username": "sdevaraju7"
}]
```

POST post/create

Create a new post in a community collection in database.

Resource URL

https://testing-coh77z5j7a-ue.a.run.app/post/create

Request Information

Request Format: JSON

Resource Information

Response Format: JSON

Requires Authentication: No

Rate Limited: No

Parameters

Parameter	Description
text	Text of the post
dateTime	Date and Time the post was created
authorId	ID of the user who created the text
communityId	ID of the community to post in
medias	Link of the images
tagged	UserID of the tagged people in post
topicId	ID of topic in Integer

Example Requests

https://testing-coh77z5j7a-ue.a.run.app/post/create

Json:

```
{
        "text": "This is the post to be created",
        "authorId": "65b7fed89cb7885873ade787",
        "communityId": "65d40edbab9c837874869dc4",
        "dateTime": "2024-04-02T01:58:41.204Z",
        "medias": "{"urls":[
        https://firebasestorage.googleapis.com/v0/b/iluvus-media.appspot.com/o/post%2Fcommu
        nity_65d40edbab9c837874869dc4%2Fuser_65b7fed89cb7885873ade787%2Filuvus65d4
        0edbab9c837874869dc465b7fed89cb7885873ade787lbb2g8xstbhchbzkklfzq3yqfi0q0h?a
        lt=media&token=8c3d95b4-624e-4152-a431-9aaf363ea943

        ]}",

        "tagged": "65d40e26ab9c837874869dc1",

        "topicId": "1"

}
```

Example Response

```
[

        {

        "id": "65d41222ddd5e6452310b841",

        "text": "Starting my day with some gentle yoga stretches and deep breathing. Nothing
beats the feeling of peace and serenity in the morning. Who else loves a good yoga session to
kickstart the day? ",

        "dateTime": "2024-02-20T02:44:44.972Z",

        "author_id": "Grand, Smith",

        "community_id": "65d40edbab9c837874869dc4",

        "report_count": null,

        "comments": [

        {

                "datetime": "2024-02-20T02:50:39.389Z",

                "text": "Whoa this is amazing!!!!",

                "id": "0d7dbbe1-9850-4486-87af-7994919286d1",

                "author_id": "65d40e3eab9c837874869dc2"

        },

        {

                "datetime": "2024-02-20T02:51:46.628Z",

                "text": "I want to do it too!!",
                "id": "2e667c76-0b14-4180-b9d4-0f9587f440f1",
                "author_id": "65d40e3eab9c837874869dc2"
        }
        ],
        "likedBy": [
        "65d40e3eab9c837874869dc2",
        "65d420bc903bd73bdd0555ed"
        ],
        "reportedBy": [],
        "tagged": null,
        "topicId": 13,
        "medias": null
        },
…
]
```

POST post/like

Uplift (like) a post and increase the number of uplifts in the database.

Resource URL

https://testing-coh77z5j7a-ue.a.run.app/post/like

Request Information

Request Format: JSON

Resource Information

Response Format: String

Requires Authentication: No

Rate Limited: No

Parameters

Parameter	Description
userId	ID of the user
postId	ID of the post to like

Example Requests

https://testing-coh77z5j7a-ue.a.run.app/post/like

Json:

```
{
        "userId": "65b7fed89cb7885873ade787",
        "postId": "660acb547ea66864247a97ea"
}
```

Example Response

“The uplift number”

“0”

POST post/report

Report a post that user thinks violate community guidelines.

Resource URL

https://testing-coh77z5j7a-ue.a.run.app/post/report

Request Information

Request Format: JSON

Resource Information

Response Format: String

Requires Authentication: No

Rate Limited: No

Parameters

Parameter	Description
userId	ID of the reporting user
postId	ID of the post to report

Example Requests

https://testing-coh77z5j7a-ue.a.run.app/post/report

Json:

```
{
        "userId": "65b7fed89cb7885873ade787",
        "postId": "660acb547ea66864247a97ea"
}
```

Example Response

“Post reported Successfully”, “Post reporting Failed”

POST community/create

Creates a community and returns status message.

Resource URL

https://testing-coh77z5j7a-ue.a.run.app/community/create

Request Information

Request Format: JSON

Resource Information

Response Format: String

Requires Authentication: No

Rate Limited: No

Parameters

Parameter	Description
name	Name of the community
description	Description of the community
rules	Rules of the community
visibility	Visibility of the community
moderator	Moderator of the community
image	Image of the community
ownerId	ID of owner of the community

Example Requests

Json:

```
{
        "description": "First Community",
        "image": "",
        "moderators": "65d40e26ab9c837874869dc1",
        "name": "Georgia Tech",
        "ownerId": "65b7fed89cb7885873ade787",
        "rules": "Be Nice!",
        "visibility": "Public"
}
```

Example Response

“Community created successfully”, "Community creation failed"