ENCOMPASS

Name	Student Number
JULIANNA VENTER	U20433752
MORGAN BENTLEY	U18103007
KEABETSWE MOTHAPO	U21543462
RONIN BROOKES	U19069686
SAMEET KESHAV	U21479373

Version 2 - June 2023

Table of Contents

Introduction

Encompass is a progressive web application (PWA) that aims to provide users with personalized recommendations for likeminded communities, friends, movies, series, novels, and educational content. It uses machine learning to analyse user data and gather content from different sources and uses that information to personalize the app content. Users can create profiles, customize preferences, search for content, connect with other users through social networking features, and access the app across different devices. The app prioritizes security, privacy, and usability, and may have optional gamification elements and social media integration.

Requirement Analysis

Requirement analysis is a crucial step in the software development process that helps to identify and document the needs and expectations of stakeholders for the software product. Here is a requirement analysis for developing the Encompass social media application:

1. **Purpose and Scope**

The Encompass social media application is intended to connect users based on common interests. It will allow users to create and join groups, share content, and connect with others who share similar interests.

2. **Objectives**

The objectives of the Encompass social media application are:

1. To provide a platform for users to connect with others who share similar interests and passions.

2. To create a user-friendly interface that allows users to easily search for and join groups based on their interests.

3. To facilitate the sharing of content, including photos, videos, and written posts, among users with common interests.

4. To promote community building and foster connections between users who might not have otherwise met.

5. To provide tools for users to organize and host events (such as quizzes and marathons), and other social gatherings with members of their groups.

6. To collect and analyze data on user behaviour and preferences in order to continuously improve the user experience and better understand user needs.

7. To ensure the privacy and security of user data and information by implementing robust security measures and adhering to data protection regulations.

3. **System Requirements**

System requirements for Encompass refer to the hardware, software, and infrastructure needed to run the application. Here are some system requirements for Encompass:

1. Hardware: Encompass should be able to run on a range of hardware configurations, including desktops, laptops, tablets, and smartphones. The

minimum hardware specifications should be defined to ensure the application can run smoothly.

2. Web browser: Encompass should be compatible with major web browsers such as Chrome, Safari, Firefox, and Microsoft Edge. The application should be designed with responsive web design to ensure proper display and functionality on various screen sizes.

3. Database management system: The application should be supported by a reliable and scalable database management system (DBMS) that can handle large amounts of user data and content. The DBMS should provide features such as backup and recovery, performance optimization, and data security.

4. Third-party integrations: Encompass may require integration with third-party services such as payment gateways, social media platforms, or cloud storage services. These integrations should be supported by the system and properly configured to ensure data security and privacy.

5. Security and data protection: Encompass should implement robust security measures to protect user data and content from unauthorized access, modification, or theft. The application should use encryption, access control, and other security protocols to ensure data protection. Regular security audits and vulnerability testing should also be conducted to identify and mitigate potential security risks.

Functional requirements

Here are some functional requirements for the development of the Encompass social media application:

1. User registration and profile creation: The application must allow users to create a profile and register for the platform, providing basic information such as their name, email, and password. Users should also be able to create a profile that includes their interests and other relevant information.

2. User Profile Management: The application should allow users to manage their profiles, including adding a profile image, bio and personal information. The users should be able to update this information and their privacy settings as well.

3. Group creation and management: Users should be able to create groups based on their interests and manage them effectively, including adding and removing members, setting group rules, and posting content. Admin roles should also be implemented to allow for moderation and group management.

4. Feed/Home Page: The application should have a dynamic home page that serves as a centralized hub for users. The home page should provide personalized feed showcasing relevant and recent content from the user's joined communities and friends. The home page should display recommendations for new communities based on the user's interests and previous activities; and should highlight trending or popular posts and discussions across all communities to encourage engagement.

5. Explore and search: The application should allow users to easily discover and search for communities based on their interests and preferences. Search filters

such as group size, location, and category should also be available. The search filter will also include profiles, posts, events and categories too.

6. Content sharing and creation: Users should be able to share and create content such as text, photos, and videos within their groups. A user-friendly editor should be provided to allow users to create and edit their content.

7. Events creation and management: Users should be able to create and manage events within their groups, including setting a date, location, and details.

8. Notifications and messaging: Users should receive notifications when new content is posted in their groups, when new events are created, and when they receive messages. Messaging functionality should also be available to allow users to communicate with each other within the platform.

9. Reporting and moderation: Users should be able to report inappropriate content and behaviour, and moderators should have the ability to remove or hide inappropriate content and users.

10. Analytics and insights: The application should be able to collect and analyse data on user behaviour, such as which groups are most popular and which content is most engaging, in order to continuously improve the user experience.

11. Security and privacy: The application should have robust security measures in place to protect user data and information. Users should also have control over their privacy settings and be able to opt out of some data collection.

12. Cross-platform compatibility: The application should be compatible with various devices and operating systems, including web browsers, iOS, and Android.

User Characteristics

Understanding the characteristics of different user types, their technical proficiency, frequency of use, interests, privacy concerns, and mobile usage patterns will help inform the design and development of Encompass to ensure a user-centric experience. Below are the user characteristics of Encompass:

- User Types:
    - General Users: These are individuals who use Encompass to join communities, share content, and engage with other users.
    - Community Administrators: These are users who have additional privileges and responsibilities within specific communities, including content moderation, user management, and enforcing community guidelines.

- Technical Proficiency:
    - Novice Users: These users may have limited experience with social media applications and may require intuitive and user-friendly interfaces.
    - Intermediate Users: These users are familiar with social media platforms and can adapt to new features and functionalities relatively quickly.
    - Advanced Users: These users are experienced in utilizing social media applications and may expect advanced features, customization options, and quick navigation.

- Frequency of Use:
  - o Regular Users: These users engage with Encompass frequently, participating in multiple communities, posting content, and interacting with other users on a regular basis.
  - o Occasional Users: These users visit Encompass less frequently, accessing the application for specific purposes or participating in specific discussions of interest.

- Interests and Activities:
  - o Diverse Interests: Users of Encompass have a wide range of interests, including movies, music, books, series, and other relevant topics.
  - o Content Creators: Some users may actively create and share content, such as reviews, recommendations, or original works related to their interests.
  - o Content Consumers: Other users primarily consume content generated by other users and engage through likes, comments, and sharing.

- Privacy Concerns:
  - o Privacy-Conscious Users: These users prioritize their privacy and expect robust privacy settings, options to control their personal information, and secure handling of their data.
  - o Open Users: These users are comfortable sharing their interests, activities, and personal information within the Encompass community, emphasizing social connections and interactions.

- Mobile Users:
  - Mobile Users: Encompass should accommodate users accessing the application through mobile devices, requiring responsive design, optimized performance, and intuitive mobile interfaces.

User Stories

- As a user, I want personalized recommendations based on my interests, so I can discover relevant content and resources easily.

- As a user, I want to connect with like-minded individuals and communities, so I can engage in meaningful discussions and collaborations.

- As a user, I want gamification features to enhance my experience and motivation, so I can enjoy a dynamic and interactive environment.

- As a user, I want the ability to share my interests and resources with others, so I can contribute to the community and foster knowledge exchange.

- As a user, I want to create a profile with my interests and preferences, so Encompass can personalize recommendations and connections tailored to my specific tastes.

- As a user, I want to explore a wide range of content, including posts, videos, communities, and events, that align with my interests, so I can stay informed and inspired.

- As a user, I want to join or create communities focused on specific topics, hobbies, or professions, so I can connect with others who share similar passions and exchange knowledge.

- As a user, I want to receive notifications or updates about new content, community activities, and relevant events, so I can stay engaged and never miss out on exciting opportunities.

- As a user, I want to participate in discussions, comment on content, and interact with other users, fostering a sense of community and collaboration.

- As a user, I want to track my progress, earn badges, and unlock achievements based on my engagement and contributions, enhancing my sense of accomplishment and motivation.

- As a user, I want to have control over my privacy settings, allowing me to manage the visibility of my profile, interests, and interactions within the Encompass community.

Use Cases

Note: <u>Underlined – Demo 1</u>. **Purple & Bold – Demo 2.**

1. **Account Management Subsystem**
    1. <u>Sign Up</u>
    2. <u>Login</u>
    3. <u>Choose Preferences</u>
    4. <u>Edit Profile</u>
    5. Sign out
    6. **View posts and comments**
    7. Change Theme

2. **Post Subsystem**
    1. **View Posts** & Recommendations
    2. **Create Posts**
    3. **Like/dislike Posts**
    4. **Comment on post/Start Thread**
    5. **Share Posts**
    6. **Report Posts**
    7. **Reply on Comment**
    8. Delete Posts
    9. Delete Comments/Threads

3. **Events Subsystem**
    1. View events
    2. Create events

3. Edit event details

4. Join/RSVP event

4. **Profile Page Subsystem**

    1. Edit Personal Information

    2. Edit Preferences/Customize Interests

    3. Personalize theme and view

    4. View badges

    5. Share Profile

5. **Notifications Subsystem**

    1. Receive notifications

    2. View notification

    3. Mark as read/unread

    4. Delete notifications

    5. Edit notification settings

6. **Search Subsystem**

    1. Filter Search

    2. Search friends

    3. Search communities

    4. Search events

    5. Search Keywords

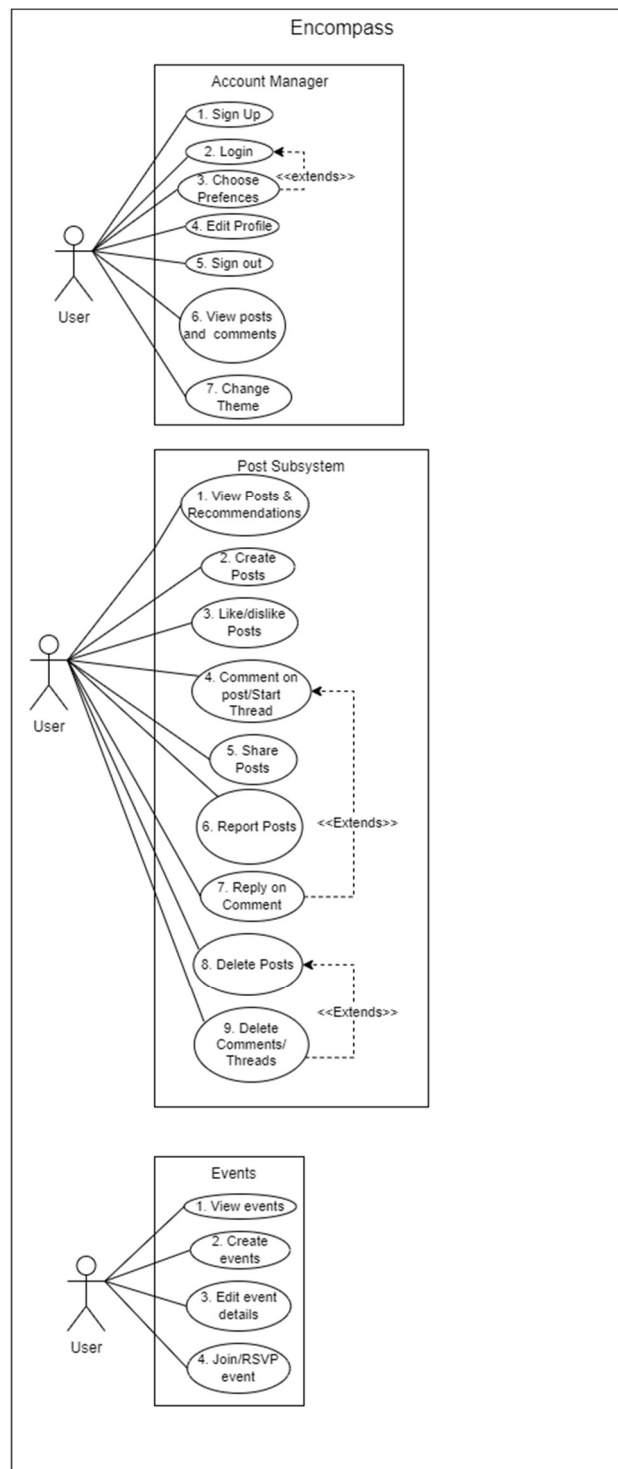   6. View results/recommendations

7. **Communities Subsystem**
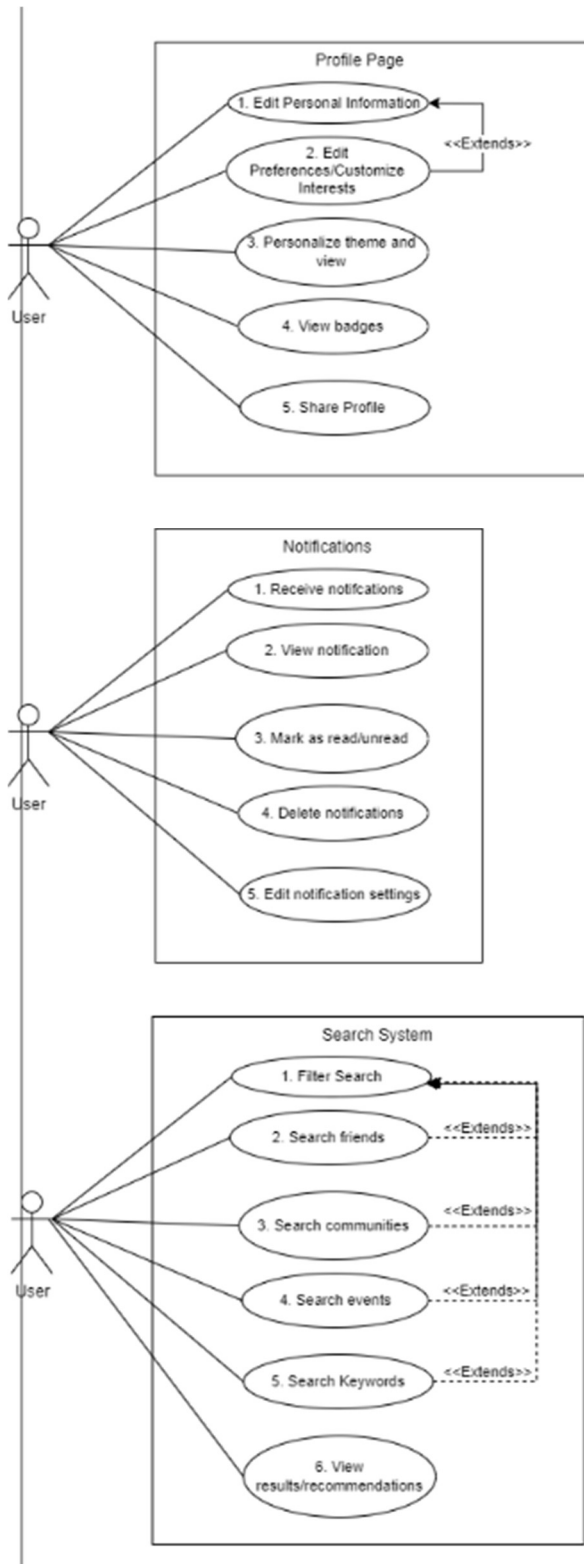
    1. View community profile

    2. **Create Community**

    3. Join Community

    4. Leave Community

    5. Assign admin

    6. Edit community profile

    7. Add/Remove members

    8. Delete community

8. **Messaging Subsystem**

    1. Start chat

    2. Send Message

    3. Receive message

    4. Delete message

    5. View chat history

Use Case Diagrams

Profile Page

1. Edit Personal Information

2. Edit Preferences/Customize Interests

<<Extends>>

3. Personalize theme and view

4. View badges

5. Share Profile

User

Notifications

1. Receive notifcations

2. View notification

3. Mark as read/unread

4. Delete notifications

5. Edit notification settings

User

Search System

1. Filter Search

2. Search friends

<<Extends>>

3. Search communities

<<Extends>>

4. Search events

<<Extends>>

5. Search Keywords

<<Extends>>

6. View results/recommendations

User

Communities

1. View community profile
2. Create Community
3. Join Community
4. Leave Community
5. Assign admin
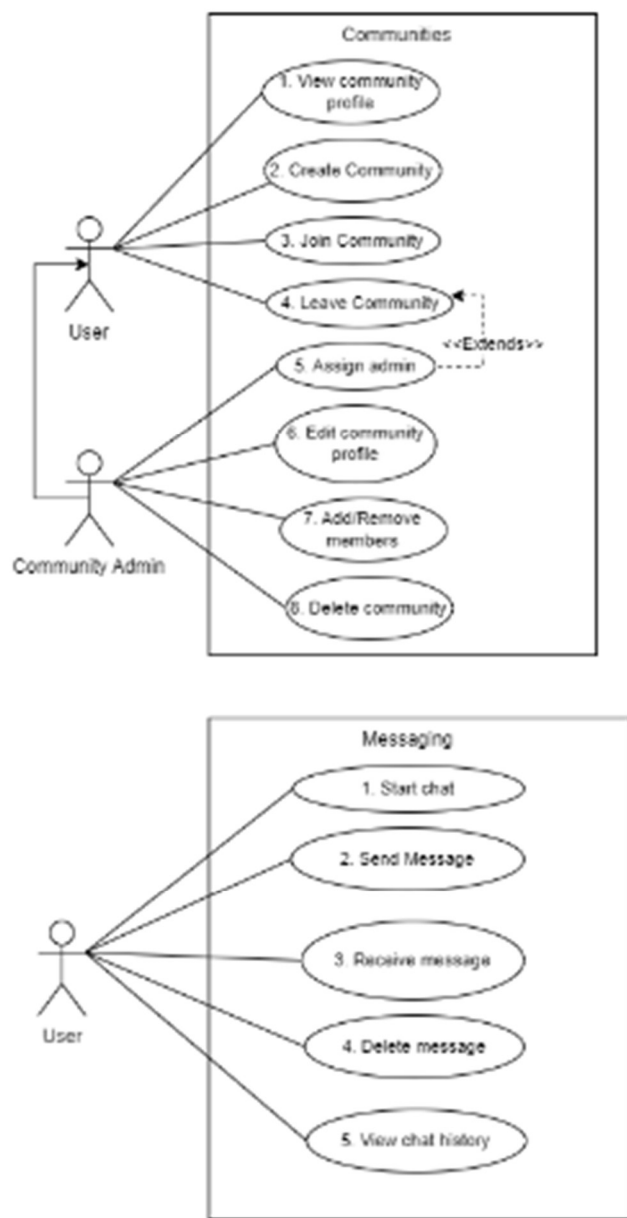6. Edit community profile
7. Add/Remove members
8. Delete community

<<Extends>>

User

Community Admin

Messaging

1. Start chat
2. Send Message
3. Receive message
4. Delete message
5. View chat history

User

Business Analysis

Encompass is a social media application that connects users based on common interests. To develop this application, a comprehensive business analysis is required to identify its potential market, competition, and costs.

1. Market Analysis: The target market for Encompass is broad, as the application aims to connect users based on their interests. However, the primary target audience is likely to be millennials and Gen Z who are already active on social media and are interested in meeting new people with similar interests. Market research should be conducted to identify the specific interests of this demographic and ensure that the application aligns with their preferences.

2. Competition Analysis: The social media market is highly competitive, with established players like Facebook, Instagram, and Twitter dominating the market. However, Encompass can differentiate itself by providing a more focused and personalized user experience that caters to users' specific interests. Competitor analysis should be conducted to identify the strengths and weaknesses of existing players and identify gaps that Encompass can fill.

3. Costs: The costs associated with developing Encompass will include software development, hosting, marketing, and maintenance costs. The software development cost will depend on the size and complexity of the application and the development team's skills and experience. Hosting and maintenance costs will depend on the application's user base and the infrastructure required to

support it. Marketing costs will depend on the target audience and the channels used to reach them.

Developing Encompass without a budget presents a unique challenge for the business analysis. In this scenario, the focus will be on identifying potential cost-saving measures and alternative resources that could be used to develop the application. Below we discuss further with a budget of ZAR 0.00:

Firstly, the development team could consider using open-source software and tools to build Encompass. This would eliminate the need for expensive software licenses and reduce development costs. Additionally, the team could leverage existing libraries and frameworks to accelerate development and reduce costs.

Another potential cost-saving measure is to rely on volunteer developers (in this case, Perfect Strangers) who are passionate about the Encompass mission and are willing to contribute their time and skills to the project. This would require effective communication and collaboration among team members, as well as a strong commitment to the project's goals and vision.

Furthermore, the Encompass team could leverage low-cost or free resources for hosting and infrastructure, such as cloud-based services like AWS Free Tier or Google Cloud Platform Free Tier. This would enable the team to build and deploy the application without incurring significant hosting costs.

Overall, developing Encompass without a budget will require the team to be creative, resourceful, and strategic in their approach. By leveraging open-source tools, volunteer resources, and low-cost infrastructure options, the Encompass

team can build a high-quality application that achieves its mission and goals, despite the absence of a significant budget.

4. Conclusion: Developing Encompass will require a comprehensive business analysis to identify its target market, competition, and costs. By conducting market and competitor analysis, and estimating the costs associated with developing and maintaining the application, the Encompass team can make informed decisions and build a successful social media platform.

Service Contracts

Services

1. **User services:** his service relates to user-related functionality such as registration, login, profile management, user settings, and user preferences.
2. **Posting services**: this service handles the creation, retrieval, updating, and deleting of a user's posts. This also involves interactions such as sharing, liking, disliking, and commenting on posts.
3. **Community services:** this service involves creating and joining, or leaving, communities, interacting with those relevant posts and events, as well as interaction with other users in that community.
4. **Social services:** this service deals with user connections, such as following and unfollowing users, and managing their friend lists.
5. **Notification services:** this service handles events such as alerting users of new messages or post interactions, as well as new followers.
6. **Messaging services:** this service enables real-time communication between users through private messages, or group chats (post comment threads that reply to a specific user(s)).

7. **Search service:** this service provides functionality for searching and discovering users, posts, categories, or events.

8. **Recommendation services:** this service is responsible for collecting user data relevant to a user's activities, post engagement, and interest expression throughout the app and it's features. This data is then used for pushing recommended content.

9. **Authentication services:** this handles user authentication and authorization, including validating credentials and generating access tokens.

10. **Media services:** this service manages uploading, storing, and retrieval of media files such as profile pictures, post images, or video content.

11. **Information services:** this service manages the uploading, storing, and retrieval of user and app-wide information responsible for populating the app media and information.

Service Interfaces

1. User Service:

- `registerUser(userDetails)`: Registers a new user with the provided user details.

- `loginUser(credentials)`: Authenticates a user with the provided credentials and generates an access token.

- `updateUserProfile(userId, updatedProfile)`: Updates the profile information of a specific user.

- `getUserSettings(userId)`: Retrieves the user settings and preferences for a specific user.

2. Posting Service:

- `createPost(userId, postContent)`: Creates a new post for the specified user with the given content.
- `getPost(postId)`: Retrieves a specific post by its ID.
- `updatePost(postId, updatedContent)`: Updates the content of a specific post.
- `deletePost(postId)`: Deletes a specific post.
- `likePost(postId)`: Likes a specific post.
- `dislikePost(postId)`: Dislikes a specific post.
- `commentOnPost(postId, commentContent)`: Adds a comment to a specific post.

3. Community Service:

- `createCommunity(userId, communityDetails)`: Creates a new community with the provided details, associating it with the specified user.
- `joinCommunity(userId, communityId)`: Allows a user to join a specific community.
- `leaveCommunity(userId, communityId)`: Removes a user from a specific community.
- `getCommunityPosts(communityId)`: Retrieves the posts associated with a specific community.
- `interactWithCommunityPost(userId, postId, interactionType)`: Performs an interaction (e.g., like, comment) on a post within a community.

4. Social Service:

- `followUser(followerId, followingId)`: Allows a user to follow another user.
- `unfollowUser(followerId, followingId)`: Removes a user from the list of followers for another user.
- `getFriends(userId)`: Retrieves the list of friends for a specific user.

5. Notification Service:

- `sendNotification(userId, notificationContent)`: Sends a notification to a specific user with the given content.
- `getUnreadNotifications(userId)`: Retrieves the unread notifications for a specific user.

6. Messaging Service:

- `sendMessage(senderId, recipientId, messageContent)`: Sends a private message from one user to another.
- `createGroupChat(members, chatDetails)`: Creates a group chat with the specified members and details.
- `sendMessageToGroupChat(chatId, senderId, messageContent)`: Sends a message to a specific group chat.

7. Search Service:

- `searchUsers(query)`: Searches for users based on the provided query.
- `searchPosts(query)`: Searches for posts based on the provided query.

- `searchCommunities(query)`: Searches for communities based on the provided query.
- `searchEvents(query)`: Searches for events based on the provided query.

8. Recommendation Service:

- `getRecommendedContent(userId)`: Retrieves recommended content for a specific user based on their activities and interests.

9. Authentication Service:

- `authenticateUser(credentials)`: Authenticates a user with the provided credentials and generates an access token.

10. Media Service:

- `uploadProfilePicture(userId, imageData)`: Uploads and sets the profile picture for a specific user.
- `uploadPostImage(postId, imageData)`: Uploads an image for a specific post.
- `uploadVideo(videoData)`: Uploads a video to the media storage.

11. Information Service:
- getAppInformation(): Retrieves app-wide information such as terms of service, privacy policy, or general information.

1. **User Services:**

User registration and profile information can be represented using JSON or a structured object format that includes fields like name, email, username, password, date of birth, and preferences.

User settings can be stored as key-value pairs or a structured format, such as JSON, containing various user preferences like notification preferences, privacy settings, and theme preferences.

2. **Posting Services:**

Posts can be represented using JSON or a structured format that includes fields like post ID, content, timestamp, user ID, and any associated media.

Interactions with posts, such as likes, dislikes, and comments, can be represented using JSON objects or structured data with relevant fields like interaction type, user ID, post ID, and timestamp.

3. **Community Services:**

Community information can be represented using JSON or a structured format that includes fields like community ID, name, description, members, and associated posts or events.

Interactions with communities, such as joining or leaving, can be represented using simple requests with community ID and user ID.

4. **Social Services**

User connections and friendships can be represented using JSON or a structured format that includes fields like user ID, friend list, follower list, and following list.

Interaction actions like following or unfollowing users can be represented using JSON requests with relevant user IDs.

5. Notification Services:

Notifications can be represented using JSON or a structured format that includes fields like notification ID, content, user ID, timestamp, and associated actions or links.

Alerts or push notifications can be sent as messages in a standardized format like JSON with fields for the recipient, message content, and timestamp.

6. Messaging Services:

Private messages and group chats can be represented using JSON or a structured format that includes fields like message ID, sender, recipients, content, and timestamp.

Comment threads on posts can be represented similarly, with additional fields for the parent post and the user(s) being replied to.

7. Search Service:

Search queries can be represented using simple text strings or JSON objects with specific fields for filtering, sorting, and searching criteria.

Search results can be returned as JSON objects or structured data containing relevant information about users, posts, categories, or events.

8. Recommendation Services:

User data for recommendation purposes can be stored in a structured format like JSON, including user ID, activity history, expressed interests, and engagement metrics.

Recommended content can be provided as JSON objects or structured data with relevant fields like content ID, type, title, description, and associated metadata.

9. Authentication Services:

Authentication requests can utilize standardized protocols like OAuth or JWT (JSON Web Tokens) that generate and exchange access tokens and refresh tokens.

User credentials for authentication can be sent securely using encrypted channels or through structured requests containing username, password, and additional authentication parameters.

10. Media Services:

Media files such as images or videos can be uploaded and stored using formats compatible with web standards (e.g., JPEG, PNG, MP4) or specific frameworks.

Media retrieval can be facilitated through URLs or structured requests specifying the media ID or associated metadata.

11. Information Services:

User and app-wide information can be stored in a structured format like JSON, including data such as app metadata, static content, or user-specific information.

Retrieval of information can be facilitated through structured requests specifying the required data or metadata.

BACKEND TODO:

Authentication and authorization: If your application requires authentication and authorization, define the mechanisms for authenticating users and the authorization rules for accessing different services or resources. Specify any required authentication headers or tokens.

Error handling: Define the error handling mechanism for service interactions. Specify how errors should be communicated, including error codes, error messages, and any additional information that can help identify and resolve issues.

Testing and validation: Use the service contract as a reference for testing and validating the behavior of your services. Ensure that each service adheres to the contract and performs as expected. Use automated tests or contract-driven testing tools to validate the contract's compliance.

Backend

Authentication and Authorization: The application will be making use of authentication and Authorization to ensure that the all the user data is safe and can only be accessed by authorized users. Authentication will take place when a user is logging in or registering. When a user logs in, their login credentials will be encrypted and a userID key will be created. This UserID key will be used throughout the application for authorization. For data authorization, in the MongoDB database we will have rules that will enforce authorization in certain collections so that only the correct (the user who is logged in) can view their data and not everyone.

Error Handling: If an error must occur in the app due to a bad request to the api, then the user will be given an informative pop up saying that an error has occurred. However, to avoid many errors from being made, all user inputs will be checked for correct syntax before they are passed along to the api. If the user has inputted something incorrectly, they will be given a pop up (or prompt) to tell them what error they made, so that they are able to go back and fix it and try again.

Testing and validation: To ensure the quality and reliability of our project, we have established a testing policy focused on unit testing. Jest has been selected as the primary tool for conducting these tests.

Jest is employed to perform comprehensive unit testing on our project. Each service undergoes thorough testing, with all functions and methods being examined. The advantage of using Jest lies in its ability to automate testing and promptly identify any failures that may arise due to changes in code or dependencies.

By leveraging Jest's automated testing capabilities, we can ensure that our project's individual components are functioning as intended and remain unaffected by any code modifications or dependencies.

Our team is committed to achieving a high level of test coverage, with a target of surpassing 70% coverage by the project's completion. This entails testing most of the project's functions and methods to establish a robust and reliable codebase.

By adhering to this testing policy, we aim to deliver a high-quality project that meets the desired standards for unit testing.

Class diagram
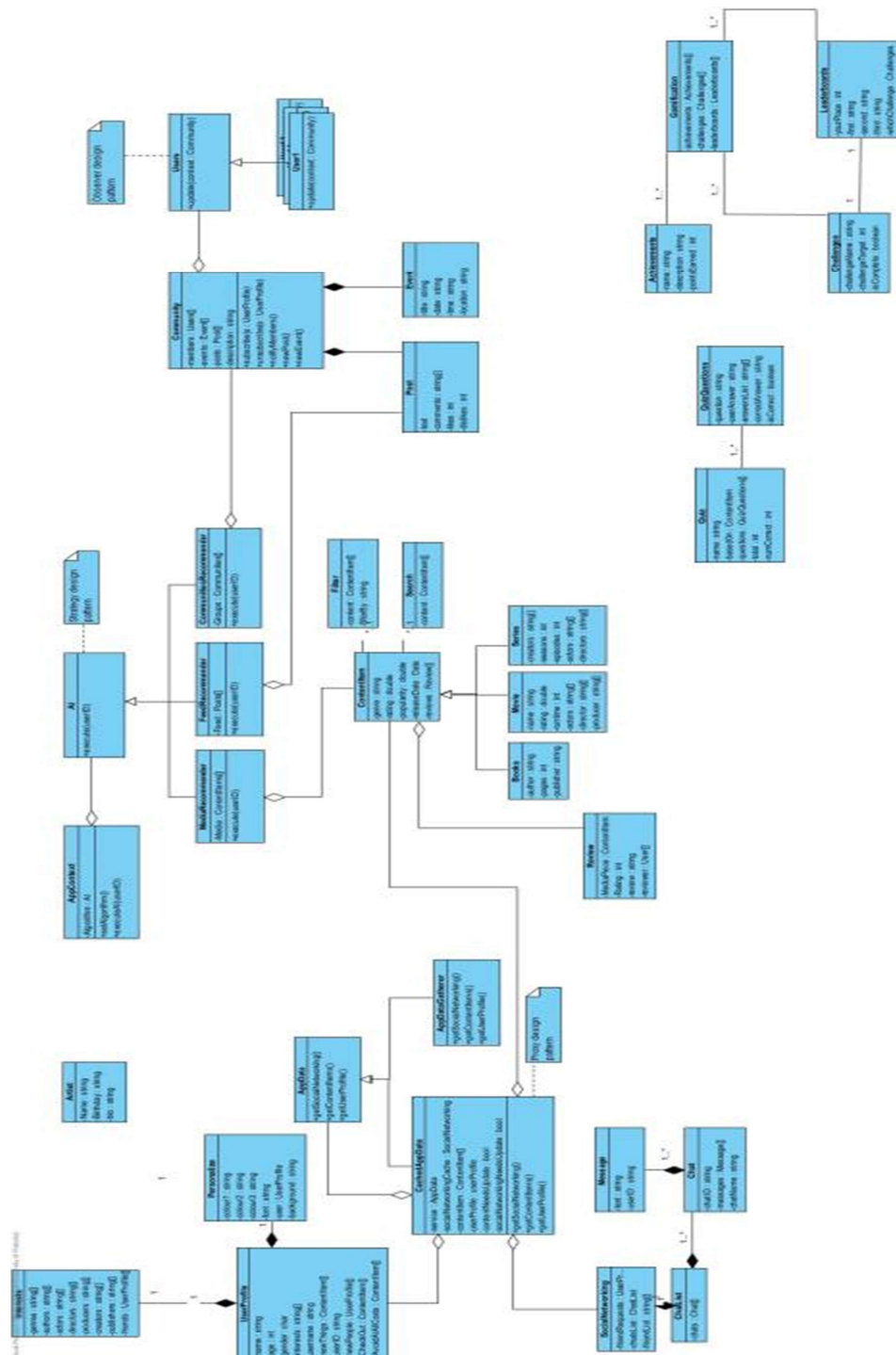
Architectural Structure

Quality requirements

Here are some non-functional requirements for the development of the Encompass social media application. Note that specifics are given in the Architectural document in our repo:

1. Performance: The application must be highly performant and able to handle a large number of users and groups without experiencing performance issues or downtime.

2. Scalability: The application should be designed to be scalable, meaning it should be able to grow and adapt to the changing needs of the user base. The application should be able to handle an increasing number of users, groups, and content without sacrificing performance.

3. Usability and user experience: The application should have a clean, intuitive, and user-friendly interface that is easy to navigate and use. Users should be able to easily find the features they need and accomplish their goals without unnecessary frustration.

4. Reliability and availability: The application must be always reliable and available to users. It should be designed to minimize the risk of outages or other interruptions in service.

5. Security: The application should have robust security measures in place to protect user data and information from unauthorized access or theft. This includes measures such as encryption, secure login, and secure data storage.

6. Compatibility: The application should be compatible with various devices and operating systems, including web browsers, iOS, and Android, and be designed to work seamlessly across all platforms.

7. Accessibility: The application should be designed to be accessible to users with visual impairment disabilities. This includes customization settings such as text size, colour contrast and themes.

8. Internationalization and localization: The application should be designed to be easily localized to support internationalization features such as time zones.

9. Maintainability: The application should be designed to be easily maintainable, with clean, modular code that is easy to modify and extend over time.

10. Performance optimization: The application should be designed to optimize performance and minimize resource usage, such as by minimizing data transfer and optimizing server-side processes.

Architectural patterns

1. Model-View-Controller (MVC)

The Model-View-Controller pattern can be utilized in Encompass to separate the application's logic into three interconnected components: the Model, the View, and the Controller. The Model represents the data and business logic of Encompass, the View handles the presentation layer for users, and the Controller manages the flow of data between the Model and the View. By adopting MVC, Encompass can achieve a clear separation of concerns, enabling easier maintenance, code reusability, and testability.

2. Multi-Tier (3 Tier)

Encompass can follow a multi-tier architecture, which involves dividing the application into three distinct tiers: presentation, application logic, and data storage. The presentation tier handles the user interface and interaction, the application logic tier contains the business logic and processing, and the data storage tier manages data persistence. By separating these tiers, Encompass can achieve scalability, maintainability, and reusability. Additionally, this architecture allows for independent updates and modifications to each tier, enhancing the overall flexibility of the application.

3. Event Driven

The Event Driven architectural pattern can be employed in Encompass to facilitate asynchronous and loosely coupled communication between various components. Events can be generated in response to user actions or system events, and other components can subscribe to these events and react accordingly. For example, when a user creates a new post, an event can be triggered, and other components, such as notifications or activity feeds, can subscribe to this event to update their respective functionalities. Event-driven architecture promotes flexibility, extensibility, and modularity within Encompass.

4. CQRS (Command Query Responsibility Segregation) - specific to a monolithic application

In a monolithic architecture, Encompass can implement the CQRS pattern to separate the read and write operations within the application. CQRS segregates the application's responsibilities into separate models for reading and writing data. The write model, also known as the command model, handles the modification and updating of data, while the read model, or the query model, is optimized for efficient data retrieval and presentation to users. By adopting CQRS, Encompass can optimize performance, scalability, and

maintainability, as it allows for different models and databases tailored to specific requirements of read and write operations.

By incorporating these architecture patterns, Encompass can achieve a well-structured, scalable, and maintainable application. The Model-View-Controller pattern ensures a clear separation of concerns, the Multi-Tier architecture provides flexibility and modularity, the Event Driven pattern enables asynchronous communication, and the CQRS pattern (specific to a monolithic application) optimizes data processing and retrieval. These patterns collectively contribute to the development of a robust and efficient social media application like Encompass.

Design patterns

1. **Strategy Design Pattern**
   a. Explanation: The Strategy Design Pattern allows for the selection of different algorithms or strategies at runtime. In the context of our chat messaging app, this pattern is used to choose the relevant AI algorithm for recommending media, groups, and posts to users.
   b. Benefits:
      - Enables dynamic algorithm selection based on runtime conditions or user preferences.
      - Promotes code reusability by encapsulating specific algorithms into separate strategy classes.
      - Facilitates easy addition or modification of algorithms without impacting the client code.
   c. Use Case: In our chat messaging app, the Strategy Design Pattern is employed to recommend media, groups, and posts to users. Based on various factors such as user preferences, behaviour, or community interactions, the appropriate AI algorithm is dynamically selected at runtime to provide personalized recommendations for each section of the homepage.

2. **Observer Design Pattern**
   a. Explanation: The Observer Design Pattern establishes a one-to-many relationship between objects, where multiple observers are notified automatically of changes in a subject. For our chat messaging app, this pattern is used to notify users who have clicked the notification button about new events and messages in a specific community.
   b. Benefits:

- Supports real-time notifications to keep users informed of updates and events.
- Facilitates loose coupling between the subject (community) and its observers (users), promoting modularity.
- Simplifies the addition or removal of observers without affecting the subject or other observers.

c. Use Case: In our chat messaging app, the Observer Design Pattern is applied to notify users who have subscribed to a specific community. When new events or messages occur within that community, the subject (community) notifies all registered observers (users who clicked the notification button), ensuring they receive real-time updates.

3. **Proxy Design Pattern**
   a. Explanation: The Proxy Design Pattern provides a surrogate or placeholder for another object, controlling access to it. In our chat messaging app, the Proxy Pattern is utilized to store and update the states of the Feed, otherProfile, and Community pages when the user performs actions such as refreshing the Feed, clicking on a profile, or accessing a community.
   b. Benefits:
      - Enhances performance by avoiding unnecessary resource loading or heavy operations until needed.
      - Provides a level of indirection for accessing objects, enabling additional functionality such as caching or lazy loading.
      - Simplifies the management of object states and updates.
   c. Use Case: In our chat messaging app, the Proxy Design Pattern is employed to store and manage the states of different pages. For example,

when a user refreshes the Feed, the Proxy for the Feed page retrieves and updates the content, minimizing unnecessary server calls. Similarly, the Proxies for otherProfile and Community pages handle their respective states, providing efficient access and updating.

4. **Factory Design Pattern**
   a. Explanation: The Factory Design Pattern provides an interface for creating objects, allowing subclasses to determine the class to instantiate. In our chat messaging app, the Factory Pattern is utilized for user creation, as we have different types of users (e.g., admin, moderator, regular user), and we want to create them without modifying the client code.
   b. Benefits:
      - Promotes loose coupling between the client code and the specific user classes, enhancing flexibility.
      - Supports the addition of new user types without modifying existing code.
      - Centralizes user creation logic, making it easier to manage and maintain.
   c. Use Case: In our chat messaging app, the Factory Design Pattern is applied to create different types of users. When a new user is registered, the factory method determines the appropriate user class based on the user type selected. This allows us to accommodate various user roles without the need to modify the client code, providing scalability and maintainability.

Conclusion:

By incorporating these design patterns into our chat messaging app, we can enhance its functionality, modularity, and maintainability. The Strategy Pattern enables suitable recommendations for context of recommendations, the Observer Pattern ensures real-time notifications, the Proxy Pattern manages object states efficiently, and the Factory Pattern facilitates flexible user creation for the different tiers of users. These patterns collectively contribute to a robust and extensible chat messaging app.

Deployment Model

The AWS S3 Bucket, MongoDB AWS S3, and AWS Lambda will be combined to create a deployment model for Encompass, providing a scalable and efficient infrastructure for hosting and accessing the application.

1. AWS S3 Bucket:

Encompass will utilize an AWS S3 Bucket to store and serve its static assets. For example, the HTML, CSS, and JavaScript files that make up the Encompass frontend can be stored in the S3 Bucket. Additionally, media files such as profile pictures, community banners, and post images will also be stored in the S3 Bucket. By hosting these files in an S3 Bucket, Encompass will benefit from AWS's content delivery network (CDN), ensuring fast and reliable access to the frontend assets for users across the globe.

Example:

Encompass's homepage (index.html), along with its associated CSS and JavaScript files, can be stored in an AWS S3 Bucket. When users access the Encompass website, these static assets are retrieved from the S3 Bucket, providing a responsive and engaging user interface.

2. MongoDB AWS S3 (accessing via provided URL):

Encompass utilizes MongoDB as its database for storing and managing application data. For larger files, such as media or document files, MongoDB will be configured to leverage AWS S3 for storage. MongoDB stores file metadata and provide URLs that allow users to directly access the files from AWS S3. This approach offloads the storage

and delivery of larger files to AWS S3, ensuring efficient data management and reducing the load on the MongoDB database.

Example:

In Encompass, users can upload and share media within communities. When a user uploads a media file, such as a an image file, the file itself is stored in AWS S3, while the relevant metadata (e.g., filename, size, upload date) is stored in the MongoDB database. When other users want to access the file, they can retrieve it via a URL provided by MongoDB, which points to the file stored in AWS S3.

3. AWS Lambda for hosting PWA:

Encompass will utilize AWS Lambda as a serverless compute service for hosting its backend and API. AWS Lambda allows us, as developers, to write and deploy serverless functions that handle specific tasks or endpoints for the application. By using Lambda, Encompass benefits from automatic scaling based on demand and not worry about managing servers or infrastructure. Functions can be written to handle user authentication, data retrieval and manipulation, and other backend operations required by Encompass.

Example:

Encompass uses AWS Lambda to handle user authentication. When a user logs in to the application, a Lambda function is triggered to verify the user's credentials and generate a JSON Web Token (JWT) for authentication. This serverless function handles the authentication logic, securely validates user credentials against the stored data in the MongoDB database, and returns the JWT to the frontend for subsequent API requests.

In summary, using AWS S3 Bucket, MongoDB AWS S3, and AWS Lambda as a deployment model for Encompass provides benefits such as scalable and efficient storage of static assets, efficient management of large files, and a serverless backend infrastructure. These examples demonstrate how Encompass leverages these services to enhance its functionality and provide a reliable and scalable user experience.

Technology Requirements

The below technologies are selected by our team from the given stack as the most suitable. However, this selection is subject to change based on the needs of the client and the project requirements.

Font-end: With good reason, Angular is one of the most widely used front-end development frameworks and we are most likely to use it. Its strong features enable developers to easily design dynamic and responsive web applications. The ability to develop reusable components that can be readily used across many projects is one of the main benefits of utilizing Angular. Angular is perfect for Encompass because it offers strong capabilities for handling data and state. Angular is a great option for any front-end development project because of its strong community support and ongoing development. Angular is the way to go if you want to build contemporary, scalable, and reliable applications such as Encompass.

Back-end: NestJS's solid and scalable architecture makes it a great choice for backend development. Because it is based on Node.js, it is extremely effective and fast. It also provides a large selection of plugins and modules that can be quickly added to the project, making it a very adaptable framework. As Encompass application expands, NestJS offers a strong basis for creating microservices that are simple to manage and scale. NestJS encourages developers to produce clear and manageable code, which leads to quicker development and fewer defects. It also places a strong emphasis on modularity and testability.

Database: MongoDB is a preferred choice for software development due to its flexible data model, scalability, and performance. Its document-oriented approach allows for easy adaptation to changing requirements, while its distributed architecture and horizontal scaling capabilities ensure high availability and efficient handling of large datasets. MongoDB's powerful querying and indexing features enhance data retrieval and manipulation, while its developer-friendly features, rich ecosystem, and active community contribute to increased productivity. Overall, MongoDB offers a comprehensive solution for modern software development, particularly for applications with diverse and evolving data needs, such as Encompass.

Hosting: Using AWS for hosting provides numerous advantages that make it a compelling choice for businesses. Firstly, AWS offers unparalleled scalability, allowing you to seamlessly accommodate fluctuating levels of traffic and data storage needs. Its elastic infrastructure enables quick provisioning and scaling of resources, ensuring optimal performance and cost-efficiency. Additionally, AWS boasts a global network of data centers, enabling you to deploy your applications closer to your target audience for reduced latency and improved user experience. The platform also provides a wide range of services and tools, including compute, storage, database, networking, and security, empowering you to build and manage highly available and secure applications. Moreover, AWS's pay-as-you-go pricing model allows you to only pay for the resources you consume, eliminating upfront costs and enabling cost optimization. With its robust security measures, reliability, extensive service offerings, and flexibility, AWS is an ideal choice for organizations seeking a scalable and reliable hosting solution for their applications.

Other Technologies

Use Case	Proposed Technology
Version Control	GitHub (required)
Team Organization	GitHub Boards
Testing	Cypress or Jest
Documentation	OverLeaf (LaTex), Google Docs, Doxygen, Markdown(GitHub)
CI-CD	GitHub Actions
IDE	Visual Studio Code