ENCOMPASS

PERFECT STRANGERS

EPI-USE LABS

value through innovation

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

Architecture Document

Contents

Design Strategy

Based on quality requirements and decomposition.

Quality requirements:

Based on determined quality requirements and chosen architectures reflect what the system needs to accomplish these requirements. These include quality attributes and non-functional requirements like scalability, usability, security, reliability, and maintainability.

Decomposition:

Based on taking a monolithic system and breaking it into smaller components, breaking the complex system into more manageable subsystems, helping with understandability and maintainability.

Architectural Patterns and Styles

Encompass, as a versatile social media application, adopts a robust architectural approach that leverages various patterns to ensure scalability, maintainability, and flexibility. The architectural patterns utilized in Encompass include Event Driven and Pub-Sub, Monolithic with CQRS, Multi-Tier, and Model-View-Controller (MVC). Each of these patterns brings its own set of benefits and addresses specific aspects of the application's design and functionality. By combining these patterns, Encompass achieves a well-structured and modular architecture that enables efficient handling of events, supports a scalable and efficient monolithic architecture with Command Query Responsibility Segregation (CQRS), separates concerns across multiple tiers, and maintains a clear separation of data, presentation, and business logic through the

Model-View-Controller paradigm. This integration of architectural patterns empowers Encompass to deliver a robust and feature-rich social media experience to its users.

Event-Driven & Pub-Sub Architecture

The event-driven architecture used within Encompass is essential to mapping out the application's real-time functionality. This is particularly important considering the need for real time updates on posts, messaging, notifications, as well as the recommendation aspect of the content. This implementation of the event-driven architecture makes use of the pub-sub architecture principals to establish communication between the components through the Message Broker. This approach ensures a more seamless real-time interactions within the system.

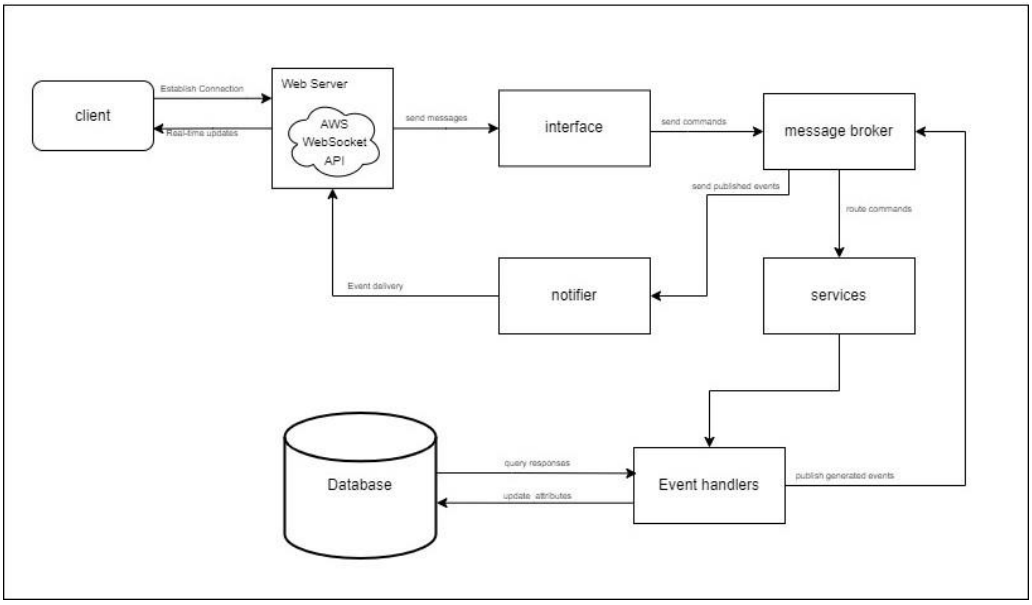
The user establishes a new connection with the websocket API, which sends the user actions or commands to the interface, which then transforms the user input into command messages. These command messages are sent to the Message broker, which acts as the central communication hub for the event-driven architecture, which then routes these commands to the appropriate application service.

Once these commands are routed to their service, they are passed to their respective event handler, which validates the command, executes the business logic of the command, and updates the system's state. Those generated events then get passed back to the message broker. The message broker then sends the notifier those published events, which in turn delivers them to the websocket.

The respective database (either key/value pair or media, depending on the nature of the event) gets updated based on the processed events, and returns relevant data for any queries.

Why use event driven?

- simplifies horizontal scalability in distributed computing models.
- More resilient to failure
- Good for real-time functionality



Monolithic & CQRS

All the components and modules of the system are tightly coupled and deployed together. This is used in combination with CQRS to divide the commands (write operations) and the queries (read operations). The system is self-contained and independent, but lacks flexibility, thus implementation of multi-tier and adding CQRS to compensate.

Commands:

- Responsible for modifying the state of the chat system.
- Processes message sending, user authentication, and other write operations.
- Writes events to the event store.

Queries:

- Responsible for retrieving data from the chat system.
- Handles message retrieval, user profile display, and other read operations.
- Reads events from the event store and generates query-specific views.

Multi-Tier

The multi-tier architecture divides the Encompass application into three distinct tiers: the presentation tier, the application/business logic tier, and the data/storage tier. This separation enables modular development, scalability, and ease of maintenance.

Presentation Tier:

The presentation tier is responsible for handling user interactions and displaying the app interface. It includes components such as the user interface (UI) and the client-side application logic.

Application/Business Logic Tier:

The application/business logic tier contains the core logic of the Encompass application. It handles user authentication, message processing, AI, and business rules. This tier communicates with the presentation tier and the data/storage tier.

Data/Storage Tier:

The data/storage tier handles the persistence and retrieval of relevant data to the Encompass app user.

Advantages:

- Introduces flexibility and reusability.
- Modularity: add or remove tiers instead or reworking entire system

Model-View-Controller

Separates the presentation layer into three interconnected components:

Model:

The Model represents the data and business logic of the application. It encapsulates the data and defines how it can be accessed, modified, or manipulated. The Model component is responsible for maintaining the integrity and consistency of the data.

View:

The View represents the presentation layer of the application. It is responsible for displaying the data to the user and providing an interface for user interaction. The

View receives data from the Model and presents it in a visually understandable format. It also sends user input or actions to the Controller.

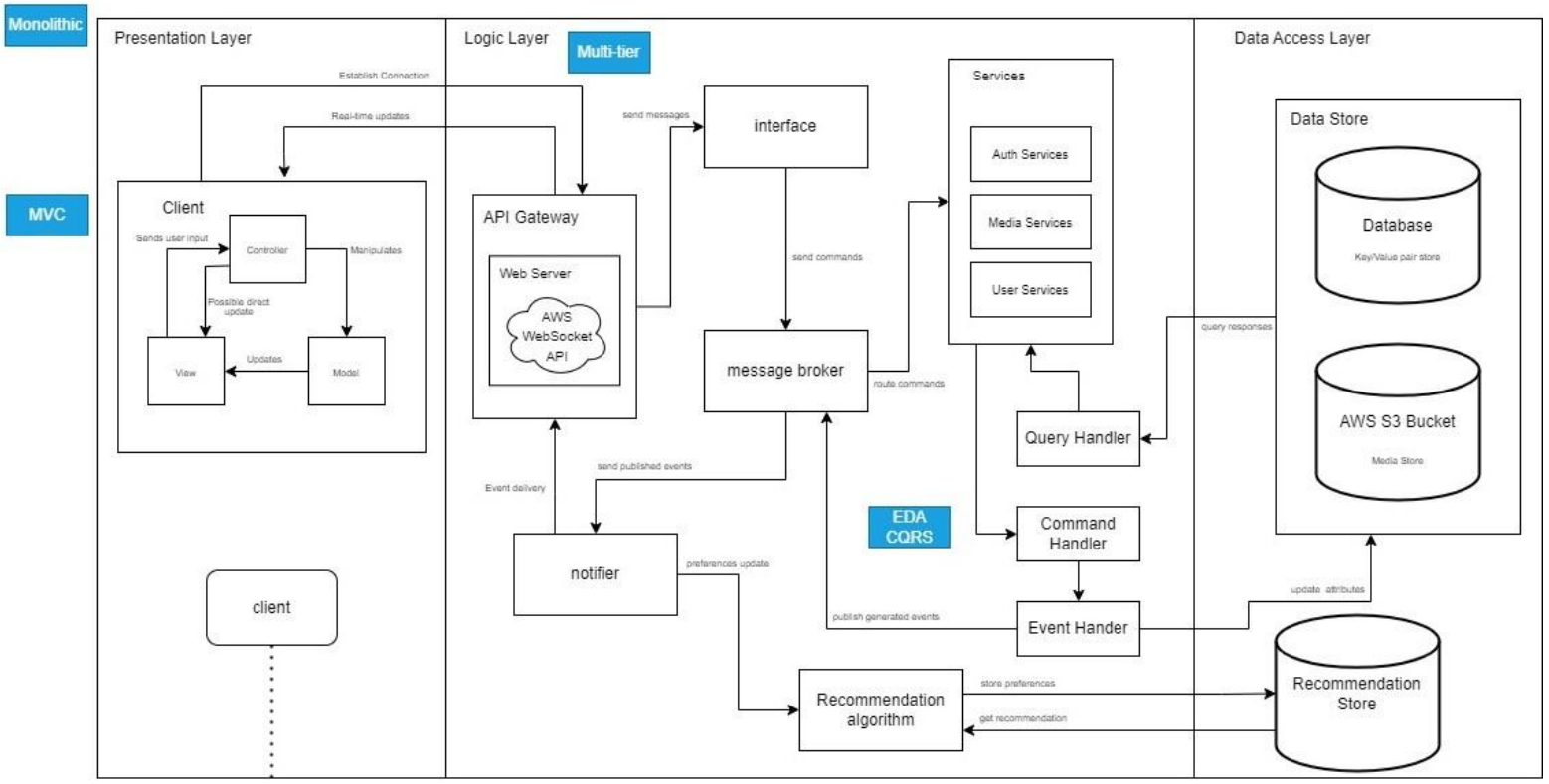
Controller:

The Controller acts as an intermediary between the Model and the View. It receives user input or actions from the View, processes them, and interacts with the Model accordingly. The Controller updates the Model based on user actions and retrieves data from the Model to update the View. It manages the flow of data and controls the overall behavior of the application.

Advantages:

- Easier maintenance, extensibility, and reusability
- Promotes loose coupling.
- Each component has its own core functionality independent of other components.
- Separates how the information is presented vs how the information is accepted by the user.

Architecture Design



Combining these strategies and patterns gives you the above design. Using the Event-driven architecture as the core structure of the system, each piece is then divided into the three Multi-Tier layers. MVC works primarily in the presentation layer, handling the user-interaction and presentation of the data to the client. CQRS and event driven structure the application logic layer, handling events and communication between sections of the system. The data access layer comprises of the databases in use for the data throughout the system, one for the gathered text-based data, one for media (stored in the AWS S3 bucket), and another for the data to be stored and used for the recommendation algorithm (this is a preliminary decision based on current knowledge and may be revised later).

Architectural Quality Requirements

Encompass places a strong emphasis on architectural quality requirements to deliver a high-performing and reliable social media platform. The application's architecture is designed with maintainability in mind, ensuring easy future development and enhancements. Scalability and performance are prioritized through efficient solutions such as horizontal scaling and optimized database queries. Security measures including secure authentication and data encryption protect user information. Usability is enhanced through intuitive interfaces and personalized experiences. Reliability is achieved through fault tolerance and automated testing. By addressing these architectural quality requirements, Encompass aims to provide a seamless, secure, and enjoyable user experience.

Maintainability

The system should remain easy to maintain throughout development and for any future iterations of the application. This would require the system to be set up in a way that makes the system easy to understand and enable developers to change or update a system component without affecting the other system components.

To quantify this requirement, we have designed the system with a modular architecture in mind, like MVC, Multi-tier, and CQRS, so that changes to any component in the respective layer or module can be added without need of in-depth understanding of or attention to the rest of the system. Another aspect would be to keep code modular and follow best coding practices. While having a physical quantity to maintainability is difficult to pin point, they can be reflected through average downtime.

Scalability and Performance

As a social media application that promotes interaction with (possible) vast communities, the concept of scalability is very important. The application needs to be able to handle an increase in users as well as their personalized data, without sacrificing the applications performance.

To quantify this requirement, modular design with the architectures mentioned in the above requirement will greatly improve ease of scalability. With the use of AWS services, a flexible option for storage is guaranteed, especially for use with visual media and the data needed for personalization algorithms. The system will be able to handle at least 20 users, with their recommendation/personalization data, before scaling up to AWS's paid tiers.

Security

As an application that deals with a lot of user data, especially gathered personal data, security is a primary concern. Architectures like Multi-tier allows for measures to be implemented at each layer.

The security requirement will be quantified using encryption and permissions. Users will need authorization and authentication through login details, and details will be encrypted before storage. This extends to non-public data as well.

Usability

The primary requirement for a social media application is ease of use. If the application lacks user-friendliness, it will inevitably result in a decline in user engagement and retention. Frustrations and negative user experiences tend to deter users from staying

actively involved with the application and its content, potentially leading to user abandonment.

Quantified in the UI/UX design to help users navigate the application without confusion or frustration, as well as interactable elements and gamification to keep attention and activity. The goal being to minimize the average amount of clicks to accomplish a task.

Reliability

As a social media application, the system must be available and operable constantly. It should also support continuous integration featuring bug fixes and updates.

This is quantifiable by its use of AWS hosting for 99.9% activity once deployed, as well as the ability to deploy new versions once updates are sent out for deployment This is because ASW has 3 availability zones, ensuring that if one fails, 2 more stand as backups until the other can be restored. This minimizes the risk of single point failure. Since AWS typically guarantees 99.9% annual uptime, the allowable downtime per year can be approximately 8 hours and 46 minutes, or 43 minutes per month maximum.

Architectural Constraints

The following constraints are imparted on the system, and solutions to handling them have been discussed in previous sections of this document:

- System must be horizontally scalable.
- Technologies used must be open source (no monetary cost)
- The system should have security measures for the verification of a user.

Technology Choices

Front-end: Angular

Overview: Angular is a popular front-end development framework that can be used to develop Encompass, a social media application. Angular offers a comprehensive set of tools and features that enable the creation of dynamic and interactive web applications. This overview will discuss the pros and cons of using Angular and provide reasons for choosing it as the development framework for Encompass.

Pros:

1. Robust and Scalable: Angular provides a structured architecture that promotes modularity and scalability. It offers features like modules, components, and services, allowing for the development of large-scale applications like Encompass without compromising maintainability.
2. Two-Way Data Binding: Angular's two-way data binding simplifies the synchronization between the application's data model and the user interface. This makes it easier to handle user interactions and update the application state in real-time.
3. TypeScript Integration: Angular is built with TypeScript, a statically typed superset of JavaScript. TypeScript enhances code quality, provides better tooling support, and enables easier debugging and refactoring. It also helps catch errors during development, leading to more reliable code.
4. Extensive Feature Set: Angular comes with a wide range of built-in features and functionalities, including form validation, dependency injection, routing, and testing tools. These features streamline the development process and reduce the need for third-party libraries.

Cons:

1. Learning Curve: Angular has a steep learning curve, especially for developers who are new to the framework or come from a different background. The extensive documentation and complex concepts may require additional time and effort to fully grasp.
2. Initial Setup Complexity: Setting up an Angular project involves multiple configurations and dependencies. This initial setup process can be complex for developers who are unfamiliar with the framework, potentially leading to delays in project kick-off.

Reasoning:

1. Component-Based Architecture: Angular's component-based architecture aligns well with the modular nature of Encompass, making it easier to manage and reuse components throughout the application.
2. Powerful Data Binding: Angular's two-way data binding simplifies the development of real-time features in Encompass, such as instant updates to posts, notifications, and user interactions.
3. Large Community and Ecosystem: Angular has a thriving community and a vast ecosystem of libraries, tools, and resources. This community support can provide valuable assistance, guidance, and readily available solutions when developing Encompass.
4. Long-Term Support and Maintenance: Angular has a history of long-term support and backward compatibility. Choosing Angular for Encompass ensures ongoing updates, bug fixes, and access to new features, reducing the risk of technology obsolescence.

In conclusion, using Angular to develop Encompass offers several advantages such as scalability, robustness, two-way data binding, TypeScript integration, and an extensive feature set. While it has a learning curve and initial setup complexity, the component-based architecture, powerful data binding, community support, and long-term maintenance make Angular a compelling choice for developing Encompass as a feature-rich and maintainable social media application.

Backend: NestJS

Overview: NestJS is a powerful backend framework that can be utilized to develop Encompass, a social media application. NestJS combines the benefits of TypeScript, Express.js, and object-oriented programming, providing a scalable and maintainable platform for building robust server-side applications. This overview will discuss the pros and cons of using NestJS and provide reasons for choosing it as the development framework for Encompass.

Pros:

1. TypeScript Integration: NestJS is built with TypeScript, which enables enhanced code quality, improved tooling support, and easier debugging. TypeScript brings static typing and interfaces to the server-side development, reducing errors and providing a more reliable codebase.
2. Modularity and Scalability: NestJS follows a modular architecture that promotes code reusability, scalability, and maintainability. The framework utilizes modules, controllers, and services to encapsulate functionality, making it easier to manage and scale the application as Encompass grows.
3. Decorator-based Programming: NestJS leverages decorators for defining routes, middleware, and other application features. Decorators simplify the creation of

endpoints and allow for centralized configuration, resulting in cleaner and more organized code.

4. Extensive Ecosystem: NestJS benefits from a vibrant and expanding ecosystem. It integrates seamlessly with various libraries, frameworks, and tools, providing developers with a wide range of options for extending the application's functionality.

Cons:

1. Learning Curve: NestJS, like any framework, has a learning curve, especially for Perfect Strangers as most of us new to the framework or come from a different background. The concepts of modules, decorators, and dependency injection may require additional time and effort to grasp fully.
2. Limited Community: While the NestJS community is growing rapidly, it may not be as extensive as some other frameworks. Finding specific resources or receiving community support may be comparatively challenging.

Reasoning:

1. Scalable and Maintainable Architecture: NestJS provides a scalable and maintainable architecture, making it ideal for building complex applications like Encompass. The modular structure and dependency injection enable efficient code organization and seamless integration of additional features as the application evolves.
2. TypeScript Benefits: TypeScript brings type safety, improved code readability, and developer productivity to the development process. By using NestJS with

TypeScript, Encompass can leverage these advantages for more reliable and efficient backend development.

3. Integration with Frontend Frameworks: NestJS pairs well with frontend frameworks like Angular, enabling a smooth development workflow between the frontend and backend. This alignment allows for seamless data exchange and shared code structures, enhancing the overall development process.
4. Robustness and Scalability: NestJS is built on top of Express.js, a highly performant and battle-tested web framework. With features like request handling, routing, and middleware support, NestJS provides a solid foundation for building scalable and robust backend services for Encompass.

In conclusion, NestJS offers numerous advantages such as TypeScript integration, modularity, decorator-based programming, and an extensive ecosystem. While it has a learning curve and a growing community, the scalability, maintainability, TypeScript benefits, integration with frontend frameworks, and overall robustness make NestJS a compelling choice for developing Encompass as a scalable and maintainable social media application backend.

Database: MongoDB

Overview: MongoDB is a popular NoSQL database that can be utilized to develop Encompass, a social media application. MongoDB's flexible and scalable nature makes it well-suited for handling large amounts of unstructured data typically found in social media platforms. This overview will discuss the pros and cons of using MongoDB and provide reasons for choosing it as the database technology for developing Encompass.

Pros:

1. Flexible Data Model: MongoDB's document-oriented data model allows for the storage of data in a flexible, schema-less format. This flexibility is beneficial for a social media application like Encompass, where data structures can vary and evolve over time.
2. Scalability and Performance: MongoDB is designed to scale horizontally, allowing Encompass to handle increased user loads and data growth. It supports sharding, replication, and auto-scaling, ensuring high performance and availability as the application expands.
3. Rich Query Language: MongoDB's query language provides powerful and expressive capabilities for querying and retrieving data. It supports various types of queries, including range queries, geospatial queries, and text search, enabling efficient data retrieval based on different criteria.
4. JSON-like Documents: MongoDB stores data in BSON (Binary JSON) format, which closely resembles JSON. This compatibility makes it easy to work with JavaScript-based frameworks and libraries, simplifying the integration of Encompass with the frontend and other components.

Cons:

1. Lack of ACID Transactions: MongoDB sacrifices ACID (Atomicity, Consistency, Isolation, Durability) transactions in favor of high scalability and performance. While it provides atomic operations on a single document, complex multi-document transactions may require careful handling and additional considerations.
2. Learning Curve: As with any new technology, there is a learning curve associated with MongoDB. Most of the Perfect Strangers developers are new to NoSQL databases and thus, may need time to understand and adapt to the document-based approach and MongoDB's specific query syntax.

Reasoning:

1. Flexible Data Structure: MongoDB's schema-less nature allows Encompass to handle evolving data structures without the need for frequent schema migrations. This flexibility accommodates the dynamic nature of social media applications where new features and data types can be introduced over time.
2. Scalability and Performance: MongoDB's horizontal scalability and high-performance capabilities make it suitable for handling the increasing user base and data volume in Encompass. The ability to distribute data across multiple servers ensures optimal performance and efficient resource utilization.
3. Data Modeling Simplicity: MongoDB's document-based data model aligns well with the natural representation of social media entities such as users, posts, and comments. The ability to embed related data within a single document eliminates the need for complex joins and enhances query performance.
4. Integration with JavaScript Ecosystem: MongoDB's support for JSON-like documents simplifies integration with JavaScript-based technologies, such as Node.js and frontend frameworks. This compatibility streamlines development and facilitates seamless data exchange between Encompass's backend and frontend components.

In conclusion, MongoDB offers several advantages such as flexible data modeling, scalability, performance, and integration with the JavaScript ecosystem. Although it lacks ACID transactions and requires a learning curve, MongoDB's strengths make it a compelling choice for developing Encompass as a scalable, flexible, and high-performing social media application backend.

Hosting: AWS

Overview: AWS (Amazon Web Services) provides a comprehensive suite of cloud computing services that can be leveraged to develop Encompass, a social media application. AWS offers a wide range of scalable and reliable infrastructure and platform services that can support the development, deployment, and management of Encompass. This overview will discuss the pros and cons of using AWS and provide reasons for choosing it as the cloud platform for developing Encompass.

Pros:

1. Scalability and Flexibility: AWS provides scalable infrastructure and services that allow Encompass to easily handle varying levels of user demand and data growth. With features like auto-scaling, load balancing, and serverless computing, AWS ensures that Encompass can scale seamlessly and efficiently.
2. Reliability and High Availability: AWS has a robust infrastructure that is designed for high availability. With multiple data centers and regions across the globe, AWS offers built-in redundancy and failover capabilities, minimizing the risk of downtime and ensuring continuous access to Encompass.
3. Wide Range of Services: AWS offers a vast selection of services that cater to various needs of Encompass, including compute, storage, database, analytics, and more. These services provide flexibility and enable developers to choose the most suitable components for building and running Encompass.
4. Security and Compliance: AWS has comprehensive security measures in place to protect Encompass and its users' data. It offers features like encryption, access

controls, identity and access management, and compliance certifications, ensuring that Encompass meets stringent security and regulatory requirements.

Cons:

1. Complexity: The extensive range of services and features offered by AWS can make it complex to navigate and configure, especially for developers who are new to the platform. Understanding and properly configuring AWS services may require additional time and expertise.
2. Cost Management: While AWS provides a pay-as-you-go pricing model, managing costs can be challenging, particularly since our budget is of ZAR0.00. Proper planning, monitoring, and cost optimization strategies are necessary to control expenses effectively.

Reasoning:

1. Scalability and Flexibility: AWS's scalability and elasticity enable Encompass to accommodate varying levels of user demand and data growth. It allows for seamless scaling and ensures that Encompass can adapt to changing requirements and user traffic patterns.
2. Reliability and Availability: AWS's robust infrastructure and global presence provide a highly reliable and available environment for Encompass. The distributed nature of AWS services reduces the risk of single points of failure and ensures high uptime and accessibility.
3. Extensive Service Offerings: AWS's wide range of services offers Encompass the flexibility to choose the most suitable tools and components for each aspect of the application. This flexibility allows developers to leverage the best services that meet Encompass's specific requirements.

4. Security and Compliance: AWS's strong security measures and compliance certifications offer peace of mind for Encompass in terms of data protection and regulatory compliance. The built-in security features and controls ensure that Encompass and its users' data are well-protected.

In conclusion, AWS offers numerous advantages such as scalability, reliability, a wide range of services, and robust security. Although it has some complexity and cost management considerations, the scalability, reliability, flexibility, and comprehensive service offerings make AWS a compelling choice for developing Encompass as a scalable, reliable, and secure social media application.