ENCOMPASS

PERFECT STRANGERS

EPI·USE LABS
value through innovation

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

Architecture Document

v3

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, 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 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.

The user establishes a new connection with the websocket API, which sends the user actions or commands 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.

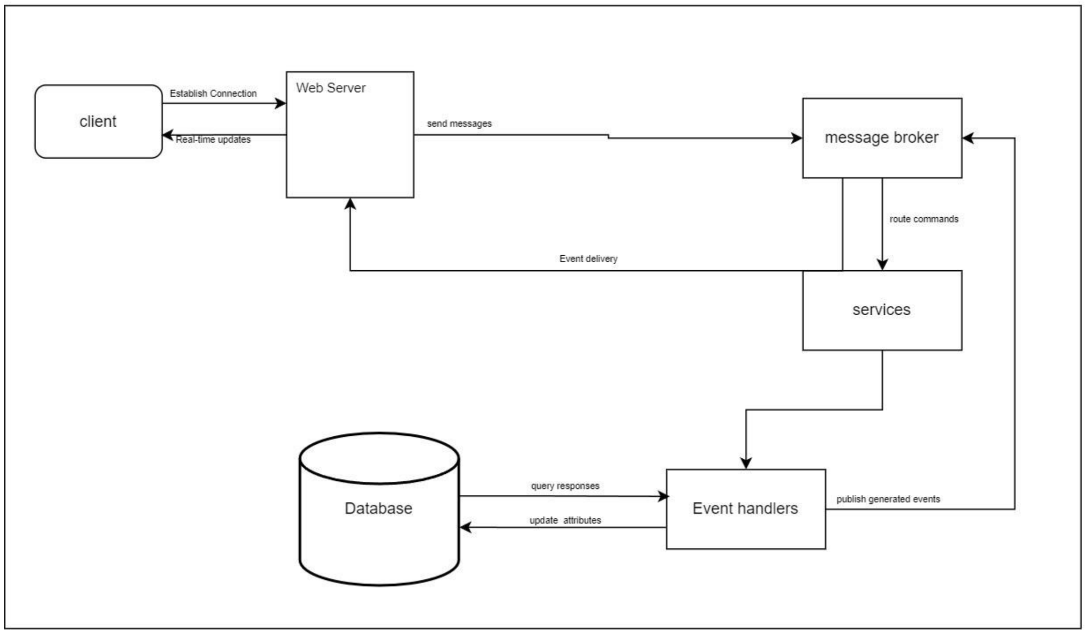
Once these commands are routed to their service, they are passed to their respective event handler, which validates the command, executes the business logic of the command, and updates the system’s state. Those generated events then get passed back to the message broker. The message broker then sends 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

<u>Model-View-Controller</u>

Separates the application logic 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.

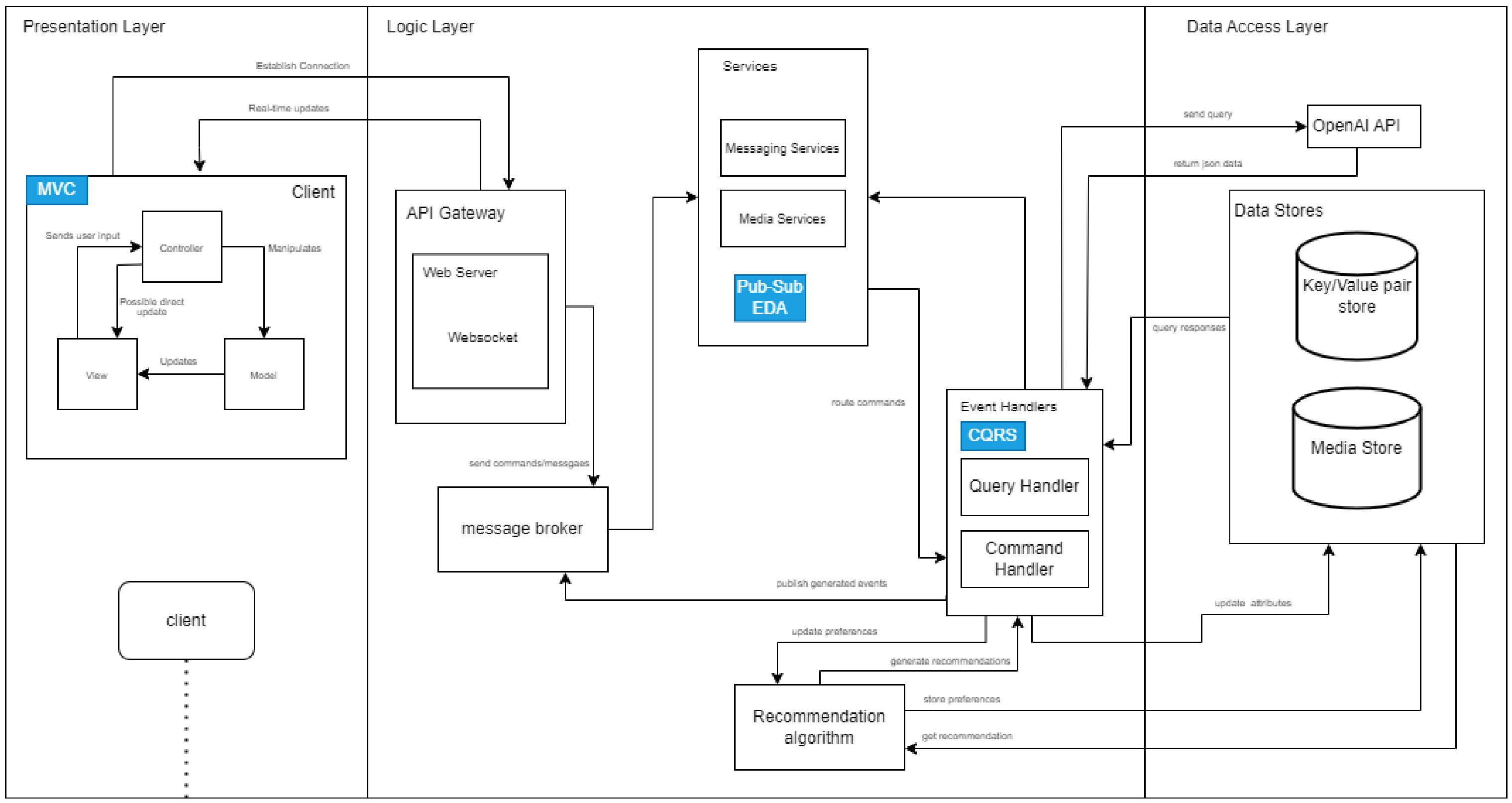
Controller:

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.



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, and one for media. The event handler also makes a query to the OpenAI API for it to respond with JSON data based on the given query context.

Architecture Design

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.

Quality Assurance:

- Code quality and standards – Please refer to the GitHub documentation
- Multi-tier and CQRS Architecture patterns
- Testing – Please refer to the GitHub documentation

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 Render services, a flexible option for storage is guaranteed.The system will be able to handle at least 20 users, with their recommendation/personalization data, before scaling up to Render’s paid tiers.

Quality Assurance:

- Event-driven architecture
- Render (hosting) resource management
    - Continuous deployment
    - Resource monitoring
    - Data log

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.

Quality Assurance:

- User authentication
- Password hashing
- Role-based Access Control (Admin rights)
- Session management - session timeout
- Render Services (DDoS and attack prevention)

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.

Quality Assurance:

- Usability testing - mentor/client, and user, feedback
  - Task Analysis and Accessibility - standards and guidelines
  - Cognitive Walkthrough - navigation evaluation

Usability testing responses (based on average rating):
Tests conducted: 13
Demographic age range: 19 – 52

Task	**Usability rating out of 5**	**Feedback**
Creating a new account	●●●●● (5)	Want more indicators for symbols and password
Navigating the feed page	●●●●●(5)	
Creating a post/event/community	●●●●○(4)	More clarity needed on keywords for events
Search	●●●●○(4)	Click space is too narrow, extend to whole card instead of just icons or name
Messaging system	●●●●●(5)	
Check and delete your notifications	●●●●●(5)	
Change your settings	●●●●○ (4.3)	Password length. Save and edit buttons were not clear at beginning. Wants button to go to themes page from settings
Change your theme	●●●●● (4.8)	Had difficulty locating sub-menu.
Join an event	●●●●●(5)	
Take a quiz	●●●●●(5)	Wants indication that quiz locks after starting
Overall navigation	●●●●●(4.7)	Wants more indicators that edited information has been saved and updated in

		the case that response time is slow.

Reliability

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

This is quantifiable by it’s use of Render hosting for 99% activity once deployed, as well as the ability to deploy new versions once updates are sent out for deployment. Render also supplies an extensive activity log of deployment status, server health, API connections, and more.

Quality Assurance:

- Incident Response Plan
    - Identify failure and communicate source to team members.
    - Assess scope of failure.
    - Isolate source (be it on a certain branch or due to outside factors, to ensure that it does not affect more parts of the system or other versions)
    - Notify those involved (if serious – communicate with clients and mentors, if able to be handled internally we alert the team member in charge of that area)
    - Research cause
    - Debug and patch
    - Test new solution
    - Make any relevant document changes
    - Communicate changes to team members and clients (if necessary)

- Comprehensive documentation

- Uptime/Downtime monitoring - Render events log
- Health check API page - Render log

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.

Pros:

1. Angular offers features like modules, components, and services, allowing for the development of large-scale applications like Encompass without compromising maintainability.

2. Angular offers two-way data binding, which improves interaction between state components and enhances the application's real-time responses.
3. Angular is integrated very will with typescript, a language which is very simple to grasp and assists with easy flaw and error detection.
4. Angular, specifically Ionic Angular, comes with a wide range of built-in libraries and component generation that can be integrated into the rest of our tech stack seamlessly.

Cons:

1. 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. 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.

Reasoning:

1. 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. 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. Angular is a very popular tool, thus it is very well documented and has a wide array of users that have published solutions and queries about it.
4. Angular is constantly updated – ensuring that Encompass's base technology will be maintained in the long term.

Backend: NestJS

Overview: NestJS is a powerful backend framework that can be utilized to develop Encompass. NestJS combines the benefits of TypeScript, Express.js, and object-oriented programming, providing a scalable and maintainable platform for building robust server-side applications.

Pros:

1. NestJS has Typescript integration, which matches perfectly with our front-end choice of Angular.
2. NestJS follows a modular architecture that promotes code reusability, scalability, and maintainability. The framework utilizes modules, controllers, and services to encapsulate functionality, making scalability easier as the application grows.
3. NestJS integrates seamlessly with various libraries, frameworks, and tools, providing developers with a wide range of options for extending the application's functionality.

Cons:

1. NestJS, like any framework, has a learning curve, especially to new developers that have not worked with it or any similar tools.
2. 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 a bit more challenging than other frameworks.

Reasoning:

1. NestJS provides a scalable and maintainable architecture, making it ideal for building complex applications like Encompass.
2. It's Typescript integration.
3. It's compatibility with our front-end Angular framework.
4. With features like request handling, routing, and middleware support, NestJS is perfect for building scalable and robust backend services.

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.

Pros:

1. The document-oriented data model used by MongoDB enables flexible, schema-free data storage. A social networking platform like Encompass, where data structures can change and advance over time, benefits from this adaptability.
2. MongoDB is designed to scale horizontally, allowing Encompass to handle increased user loads and data growth.
3. 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.
2. As with any new technology, there is a learning curve associated with MongoDB.

Reasoning:

1. The flexible data structure.
2. The scalability and performance benefits.
3. The simple data modelling structures.
4. The integration with JavaScript and equivalent languages.

Hosting: Render

Overview: Render provides a comprehensive suite of cloud computing services that can be leveraged to develop Encompass. Render offers a wide range of scalable and reliable infrastructure and platform services that can support the development, deployment, and management of Encompass.

Pros:

1. Render provides scalable infrastructure and services that allow Encompass to easily handle varying levels of user demand and data growth.
2. Render offers built-in redundancy and failover capabilities, minimizing the risk of downtime and ensuring continuous access to Encompass.

3. Render provides a wide array of monitoring capabilities, such as deployment status updates, server health, API connections status, and more
4. Render has comprehensive security measures in place to protect user data. Other than monitoring logs, it provides basic attack prevention as well as DDoS attack prevention via Cloudflare.

Cons:

1. Render has many different options to choose from in terms of services. Narrowing it down and making the decision on which of these services are the best fit for your application can be quite complex.

Reasoning:

1. Render's security measures are exactly what we need for our application, without introducing any complex systems.
2. Render has a very detailed activity log that allows us to monitor our application extensively.
3. Render has a very good basic free tier that is perfect for our needs, and if our application outgrows the constraints of the tier, there is also a pay-as-you-go option that can accommodate our growing platform.

PERFECT STRANGERS