Software Development
Group A Lyngby
10/10/2024

Development of Large Scale Systems

Assignment 3

Name	Mail
Andreas Fritzbøger	cph-af167@cphbusiness.dk
Owais Dashti	cph-od42@cphbusiness.dk

cphbusiness
COPENHAGEN BUSINESS ACADEMY

Table of Contents

Introduction

This report explores the architectural decisions behind MyDRTV, a platform designed to showcase Danish TV and films to a global audience, integrating modern social network features. By analyzing MyDRTV's architectural components, the report will outline the key technology choices and system structures that enable high availability, user interaction, and compliance with privacy regulations. The following sections offer a detailed examination of MyDRTV's system design, exploring its architecture in terms of functionality, data flow, and communication strategies.

Company Overview

DR (Danmarks Radio), also known as Denmark's state broadcaster, is a well-established media company based in Copenhagen, Denmark. With a rich history in producing and broadcasting Danish TV and radio content, DR has been a cornerstone in Danish households for decades. As part of its digital transformation strategy, DR aims to expand its reach beyond national borders by offering MyDRTV, a new digital platform focused on global audiences. MyDRTV provides users with access to a vast collection of classic Danish TV shows and films, promoting Danish culture through an engaging and interactive platform. The platform is designed to offer personalized recommendations, user ratings, and advanced search capabilities, all while ensuring compliance with stringent data privacy regulations such as GDPR. DR's commitment to innovation and cultural promotion drives the creation of MyDRTV, a social network that connects users worldwide with Danish media content.

MyDRTV requirements

We conducted an analysis to distill the essential requirements for MyDRTV. The key aspects were identified through this process to ensure the project aligns with its scope and objectives, providing a foundation for an effective development process. The requirements focus on delivering a high-quality experience, seamless video playback, and integration with social interaction features. The essential data flows cover:

Video Streaming and Playback

- Ensuring smooth playback of videos on demand for users.

- Supporting adaptive streaming to accommodate different network conditions and device capabilities.

User Engagement

- Facilitating user interaction through commenting, liking, and sharing videos on social media platforms.
- Enabling a personalized video recommendation system based on viewing history and preferences.

Security and Performance

- Guaranteeing secure video access and protecting user data with appropriate security measures.
- Ensuring the system can handle high traffic during peak times without performance degradation.

Exploring components

We've created an event storming session in which we used *Judith Birmosers* Event Storming template. These are our results (link):

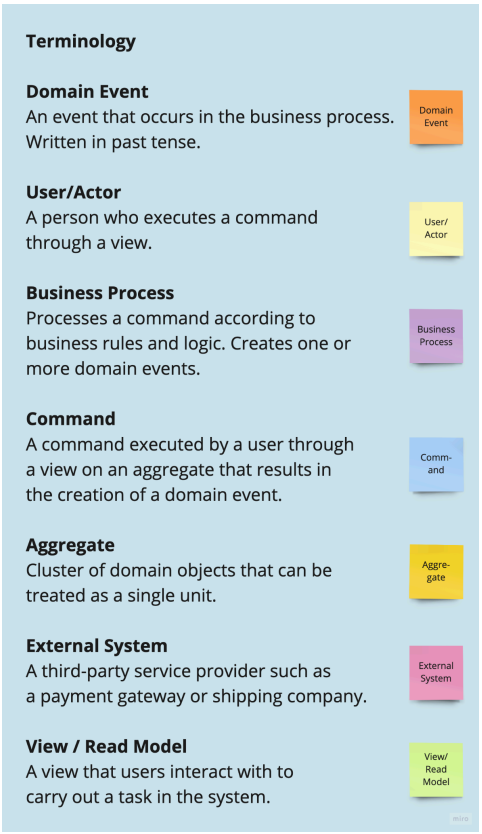
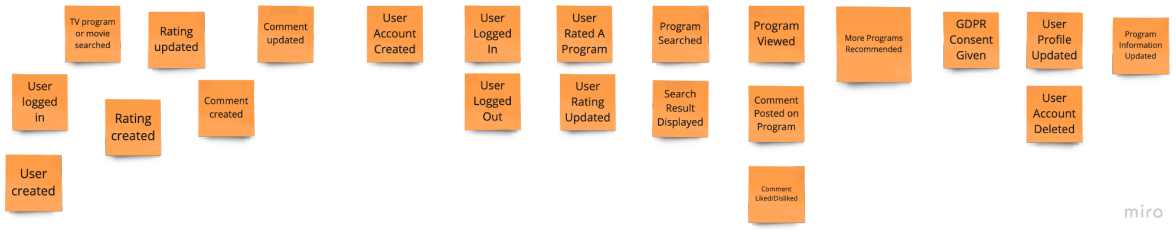


Figure 1: Key terminology applied in our event storming process[1]

Step 1: Collecting Domain Events

In this step, orange Post-Its are used to capture all the relevant domain events in the MyDRTV system. These represent key business events that have occurred, such as *CommentCreated*, *ProgramSearched*, and *GDPR_ConsentGiven*. Each event is later placed in chronological order, laying the groundwork for understanding the flow of events through the rental process.



Step 2: Refining Domain Events

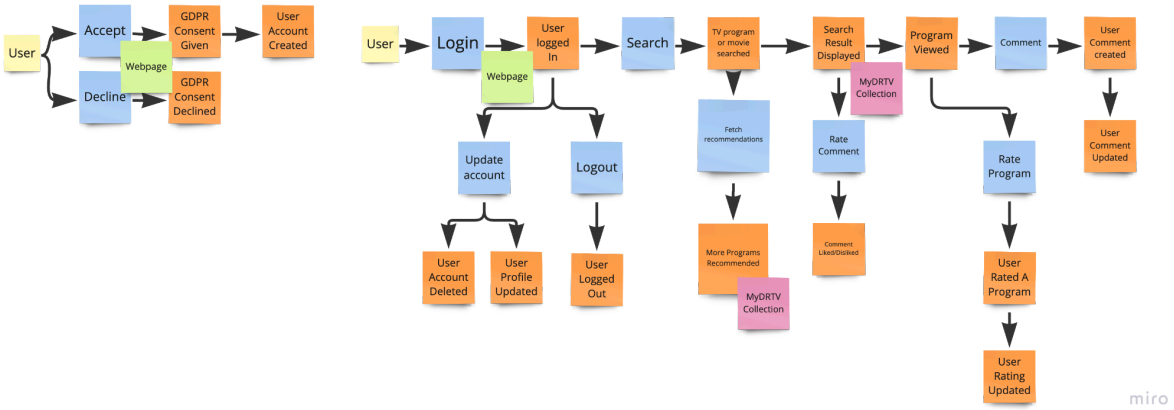
During this step, the domain events from step 1 are refined. We reviewed the events in the team to ensure that there are no duplicates, syntactically correct, meaningful, and in the correct order. This step ensures that all events are clearly defined and properly aligned with the business process.

Step 3: Tracking Causes

In step 3, the focus shifts to understanding the cases behind each domain event. Events are categorized based on their triggers, whether they are caused by user

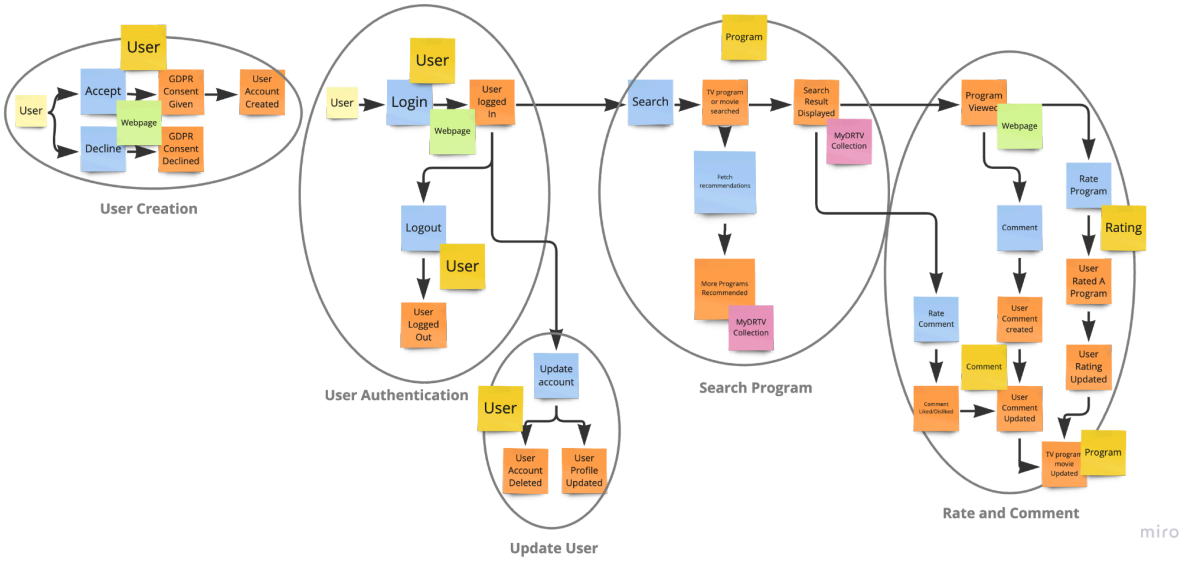
[1] Terminology defined by Judith Birmisor: Event Storming Template

actions, external systems, or other domain events. This step helps map out the relationships between events and their triggers.



Step 4: Finding aggregates & Re-sorting them

The final step involves grouping the domain events around the aggregates they impact. Aggregates such as *User, Program, Rating* are identified, and events are clustered according to their relevance. For example, *Rate program* affects the *Rating* and *Program* aggregates.



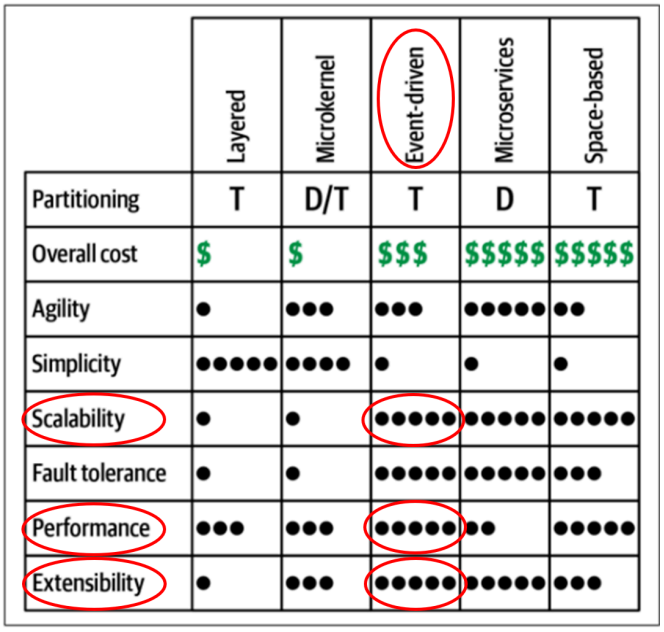
Identifying architecture characteristics

From what MyDRTV tells us, with what requirements they wish for their system, we can point out prominent architecture characteristics.
We have found three, which are the most prominent based on the MyDRTV scenario.

The system needs to be available globally, this means there is the potential for a huge amount of users registering into the system. Scalability is a system's ability to handle growth, which can be an increase of users, data, or load. A scalable architecture can handle increased demand expanding resources, horizontally, by getting more servers, or vertically, by upgrading the existing servers.
In the scenario, MyDRTV mentions specifically that they wished the system to have high availability. Performance measures how effective the system responds to user interactions or processes data. With a high performance, the system ensures quick response times and smooth user experiences.
Finally we are told, the system is part of a digital transformation project. The system needs to be able to integrate with other systems and functions. Extensibility is about how easy it is to add new functions or integrate with other systems, without concerning the entire architecture. A high extensibility makes it easier in the long-term to integrate external tools and services.
The three architecture characteristics we have identified are: Scalability, Performance, and Extensibility.

Choosing the architecture

We identified the three most prominent architecture characteristics to be: Scalability, Performance, and Extensibility.
Below is a figure rating different architectures on the characteristics. More dots indicate how high the measurement for characteristic is in the architecture, compared to other architectures. The dollar signs indicate how much more expensive the architecture is, compared to the others.



	Layered	Microkernel	Event-driven	Microservices	Space-based
Partitioning	T	D/T	T	D	T
Overall cost	$	$	$$$	$$$$$	$$$$$
Agility	●	●●●	●●●	●●●●●	●●
Simplicity	●●●●●	●●●●	●	●	●
Scalability	●	●	●●●●●	●●●●●	●●●●●
Fault tolerance	●	●	●●●●●	●●●●●	●●●
Performance	●●●	●●●	●●●●●	●●	●●●●●
Extensibility	●	●●●	●●●●●	●●●●●	●●●

Figure A-1. Architecture styles rating summary [2]

By analyzing the model, we can conclude that the event-driven architecture fits best for MyDRTV's system.
Microservices could also have been a candidate, however, the performance is not as prominent.
Space-based architecture is a close second. Event-driven does have higher extensibility than space-based has.
We can also see that event-driven's overall cost is smaller than microservices and space-based are, which is great for the company to spend less money on the system development.

An event-driven architecture is build up with the components:

- Event processor, which is usually a service, and the main deployment unit in the architecture.
- Initiative event, which usually happens outside the main system and starts a form of asynchronous workflow or process.
- Processing event, which generates, when state from a service changes, and the service reports to the rest of the system, what the state change was.
- Event channel, which is a physical message artifact used to store starting events and send them to a service, which reacts to these events.

[2] Software Architecture Patterns (Second Edition) - Mark Richards

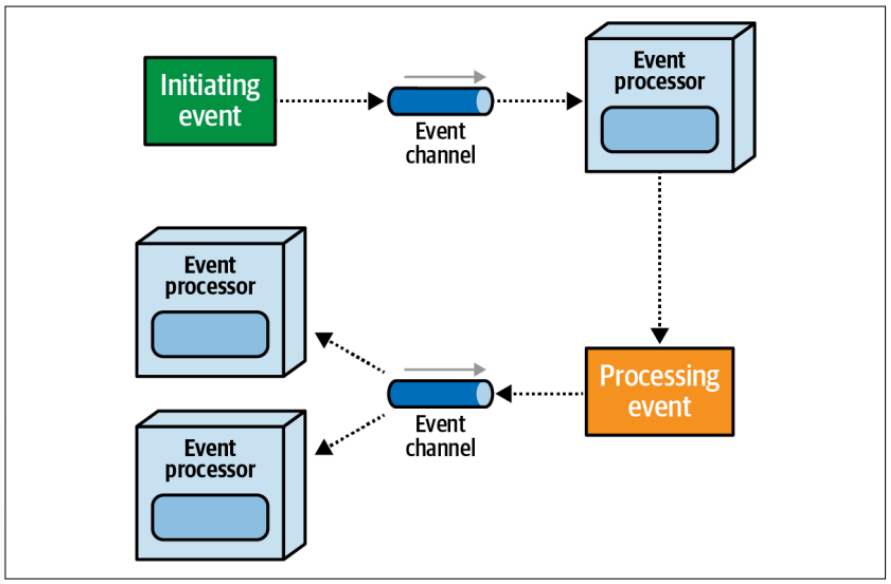


Figure 5-1. The main components of event-driven architecture [3]

The architecture is great for systems, which reacts to things happening inside and around the system, instead of responding back to the user requests. The system reacts on what the request did, and not what the request was.
Event-driven is not optimal, if you want to be in control over the workflow and the timing of the events.

Based on the requirements MyDRTV has defined in the scenario, we decide, event-driven architecture is chosen as the system's architecture.

Designing the architecture

Domain Driven Design: Strategic Design

Ubiquitous language

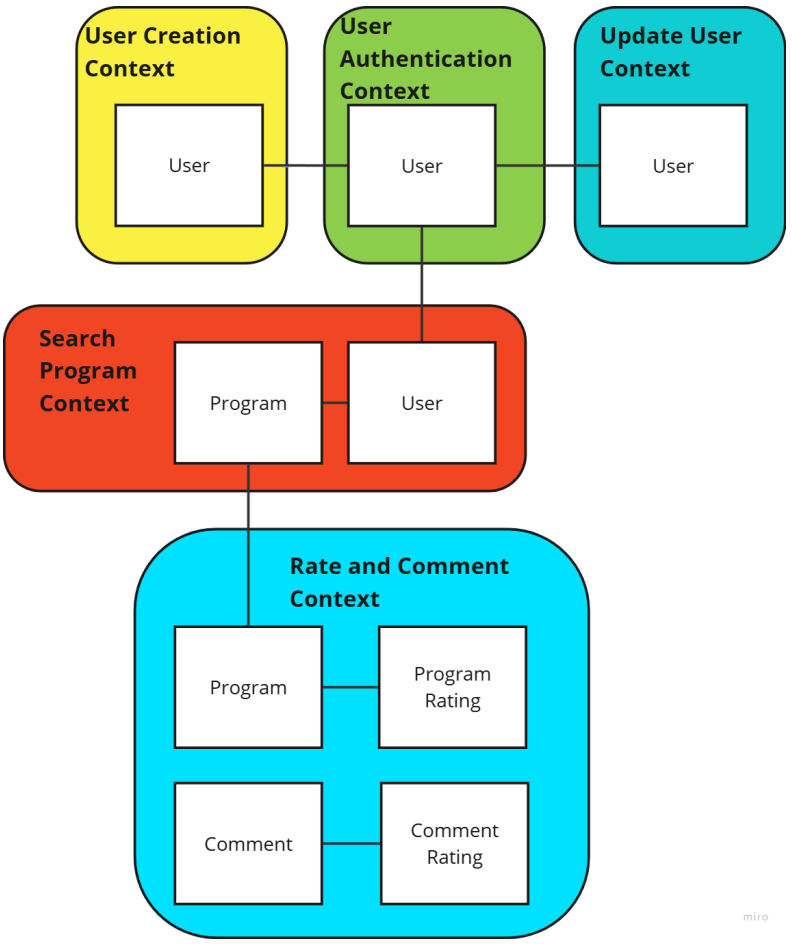
- **User:** An individual who registers on the MyDRTV platform to access features like program viewing, rating, and commenting.
- **Registration:** The process through which a new user creates an account on the MyDRTV platform by providing personal information like username, email, and password.
- **Login:** The action a user takes to authenticate themselves and gain access to their account by entering valid credentials.

[3] Software Architecture Patterns (Second Edition) - Mark Richards

- **Session:** A period during which a user is logged into the system, validated by the presence of an authentication token.
- **Video Content:** Any media file (e.g. TV shows, movies, or live broadcasts) streamed through the MyDRTV platform.
- **Channel:** A collection of video content, typically categorized by theme or broadcasting network (e.g. news, sports, entertainment).
- **Playback:** The act of streaming a video file for the viewer to watch.
- **Recommendation:** Suggested video content based on viewer preferences, watching history, or trending content.
- **On-Demand:** Video content available for streaming at any time, not bound to a live schedule.
- **Watchlist:** A personalized list of videos that viewers can save for future viewing.
- **Viewer Profile:** A record of viewer's preferences, viewing history, used to tailor the platform experience.
- **Content Provider:** External studios or broadcasters that supply video content to the MyDRTV platform.
- **Streaming Quality:** The resolution and bitrate of the video being played, which can vary depending on the viewer's internet connection and device capabilities.
- **Social Integration:** Features that allow viewers to share video content or their viewing activity on social media platforms.
- **Device Compatibility:** The range of devices (e.g. smartphones, tablets, TVs) that support MyDRTV streaming.
- **Comment:** Written feedback left by a user after watching a program.

Bounded contexts

The following are identified bounded contexts:



User Creation Context

Manages the creation of user profiles on the MyDRTV platform. Its responsibilities include:

- Handling the registration process for new users.
- Validating and storing user details (e.g. username, email).
- Ensuring unique identification of each user in the system.

User Authentication Context

Handles user login, logout and auth tokens. Its responsibilities include:

- Managing user login credentials (username, pass).
- Issuing auth tokens for secure access to the platform.
- Validating auth tokens for each request.

User Update Context

Manages updates to existing user profiles. Its responsibilities include:

- Handling user requests to update profile information (e.g. email, password, display name)
- Validating updated information and ensuring data consistency.

Search Program Context

Enables users to search for available programs on MyDRTV. Its responsibilities include:

- Managing search queries and filtering options (e.g. by genre, release date etc.).
- Returning relevant program results based on user search input.

Rate & Comment Context

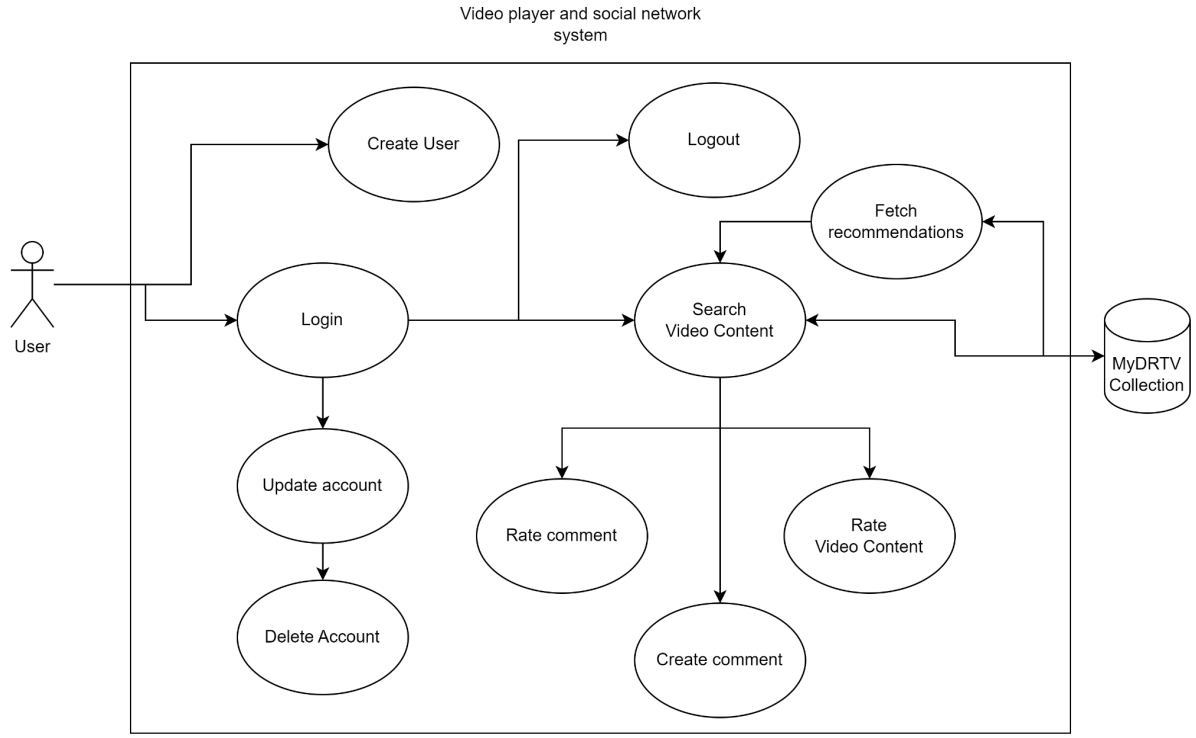
Allows users to rate and comment on programs they have watched. Its responsibilities include:

- Capturing user feedback in the form of ratings and comments.
- Storing and displaying ratings and comments alongside each program.
- Moderating comments to ensure compliance with community guidelines.

Context maps

- **User Creation Context** interacts with **User Authentication Context** to ensure that newly created users can log in immediately after registration.
    - *User Creation* sends registration details to *User Authentication,* which in turn generates a session for the user.
- **User Authentication Context** interacts with the **User Update Context** to ensure that any changes to user credentials (e.g password changes) are handled and updated in authentication records.
    - *User Update* triggers updates in *User Authentication* to ensure credentials are updated accordingly.
- **Search Program Context** interacts with the **Rate & Comment Context** to display user ratings and comments alongside program search results.
    - When a user searches for a program, *Search Program* queries *Rate & Comment* to fetch any available feedback data.

Use case diagram

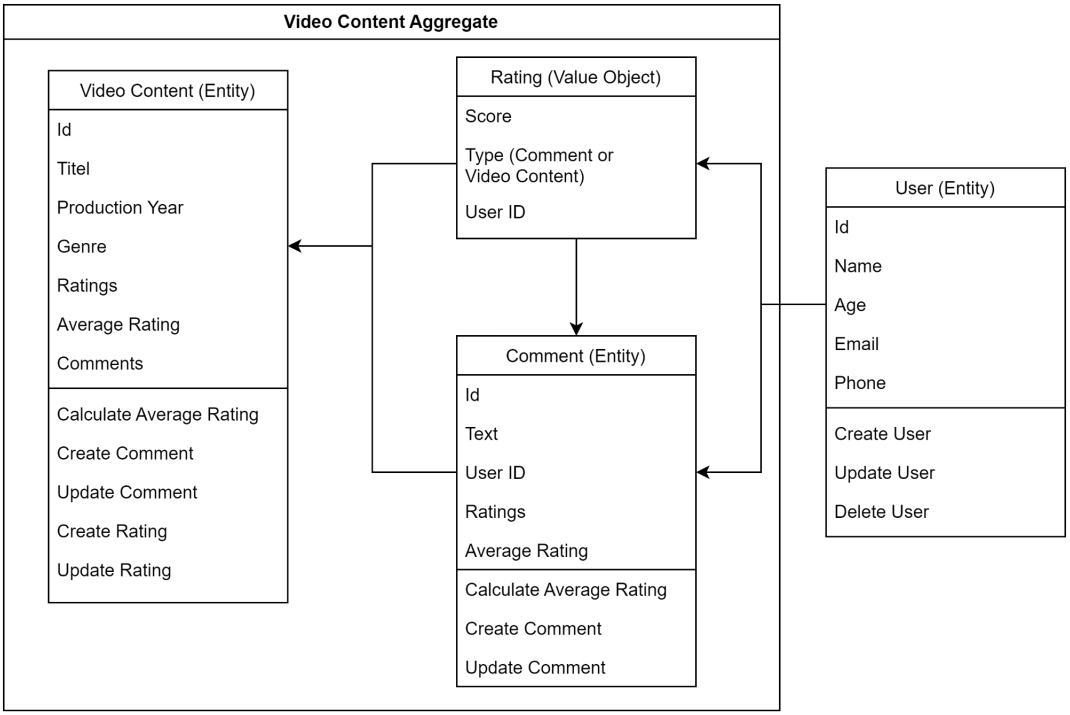


Our use case diagram shows how MyDRTV's users can interact with the system. The diagrams help to show what functionalities a user must take, before another can be executed.

The first two functions a user can do is either create a user account or login. If users want to do anymore, they would have to login first.

The function to search for video content should fetch the data from MyDRTV's collection, which can be shown to the far right of the diagram. From the collection, the data is sent back to the search function, to show the result from the search. The data storage can also send data to the function for finding recommendations, which should transform the data and send it to the result in the search function.

Domain Driven Design: Tactical Design



Domain objects → Entities & Value Objects

Entities

Entities correspond to real-world objects. They are mutable and represent key concepts in the domain. They have a unique identifier.
We have defined three entities in the domain mode, the first is Video Content, which represents movies and other tv programs MyDRTV has in their collection.
The second entity we have is User, which are the users MyDRTV wants to interact with the system.
The last entity is Comment, which are comments written by users and added to Video Content entities.
All entities have been given an Id attribute, which is the unique identifiers used to define them each.

Value Objects

Value objects represent values in our domain. They are immutable--once created, they cannot be changed. If we want a value to be something else, we need to create a new one in its place.
We have one value object, Rating, which is a value used for the Video Content Entity and Comment Entity.
Comment could have been a value object, but because we wanted users to be able to rate each other's comments, it would mean ratings for a comment can change, and would not make a Comment Entity immutable.

Aggregates

An aggregate is a cluster of domain objects that can be treated as a single unit. Each aggregate has one of its component objects as the aggregate root, from which any references from outside the aggregate should only go to. The aggregate integrity is better ensured this way.
We use aggregates, if we want to encapsulate entities and value objects and deem them as a single unit.
The aggregates we have are:

- **Video Content Aggregate**
    - Consists of: Video Content Entity, Comment Entity, and Rating Value Object.
    - Manages the lifecycle of a video program, including adding or updating ratings and comments.
    - Video Content Entity is the aggregate root.
- **User Aggregate**
    - Consists of: User Entity
    - Ensures that any user-related changes are handled through the user entity.
    - User Entity is the aggregate root.

User aggregate only consists of one domain, but by having the aggregate, the program can implement value objects or other entities in the future, which only associate to User Entity.

Repositories

- **User Repository**
    - Manages access to user data.
    - *Methods*
        - *findById(user_id)*: Retrieves a customer by their unique identifier.
        - *save(customer)*: Persists a customer entity.
        - *findByEmail(email)*: Retrieves a user by their email address,
        - findAll(): Retrieves all customers
- **Video Content Repository**
    - Handles data access for video content entities, including ratings and comments.
    - *Methods*
        - *findById(programId)*
        - *save(program)*
        - *findBySearchQuery(query, filters)*: find programs based on search query and filters.
- **Rating & Comment Repository**
    - Manages ratings and comments related to programs.
    - *Methods*
        - *addRating(programId, rating)*
        - *addComment(programId, comment)*
        - *findByProgramId(programId)*: retrieves all ratings and comments for a specific program.

Services

- **Authentication Service**
    - Handles authenticating users.
- **Search Program Service**
    - Handles logic related to program searches.
    - Also handles logic for finding recommendations for users.

Tech Stack Suggestion for MyDRTV

- **Client (Frontend)**
  - React.js (using TypeScript) via Next.js
    - Next.js for server-side rendering, static site generation, and improved SEO.
    - Redux or React query for state management
    - Tailwind CSS for styling flexibility
    - Axios for handling API requests to the backend
- **Backend**
  - Node.js
    - Non-blocking, event-driven runtime for building scalable, asynchronous applications. Ideal for handling high throughput in distributed systems.
  - NestJS (using TypeScript)
    - A progressive Node.js framework for building efficient, reliable, and scalable server-side applications.
    - Dependency injection, modular architecture, and in-built support for microservices.
    - Ideal for building an event-driven, distributed architecture.
  - Express.js
    - Used within NestJS for handling simple RESTful services. It provides minimal overhead for routing and middleware.
  - RabbitMQ or Apache Kafka
    - RabbitMQ for message queuing, allowing services to communicate asynchronously.
    - Kafka for high-throughput event streaming and real-time data processing, enabling the system to handle events such as video uploads, rating changes, and comment updates.
- **Database**
  - PostgreSQL (Relational DB)
  - Redis (In memory data store)
    - Used for caching frequently accessed data, such as video metadata, to improve the performance of high-traffic queries.
    - Also used for handling sessions

- **API Gateway**
- **DevOps/Infrastructure**
    - Docker & Kubernetes
        - Docker for containerizing services and ensuring consistency across different environments.
        - Kubernetes for orchestrating containers, enabling scaling, and managing deployments in an event-driven architecture.
    - CI/CD with GitHub Actions
        - Automated testing, building, and deploying code to ensure a smooth delivery pipeline.
- **Monitoring and Logging**
    - Grafana
        - for monitoring the health and performance of microservices.
        - for visualizing metrics and logs to monitor the system in real-time.
- **Testing**
    - Vitest / Jest
        - Unit testing
        - Integration testing
    - Cypress
        - End to end testing
    - Artillery (https://www.artillery.io/)
        - Load testing

Reflections

By using Event Storming, analyzing architecture characteristics, and developing with Domain Driven Design, we have gained an understanding for MyDRTV's requirements to the system, the system flow, and what architecture would be a best fit. Our models were helpful to get a better overview for what the domains were and what type they are, Entity or Value Object. In the event storming we defined 4 aggregates, but when we used DDD, the number got cut down to 2, because that made more sense. We think the model is ready for the software development process.