Big Systems
Ola 3
System Design and architecture

By Kristofer, Mads, Michael and Søren

Business Case:

We will go over briefly what the business case contains and how it affects our choices.

From the description of MyDRTV we get information of the economical aspects of the project described in the scenario. We are giving this information because the project wants to include global outsourcing and therefore we are giving a table of regions and the potential cost of junior, middle and senior developers from those regions.

The scenario goes on to describe the functional requirements for the application and how some of those requirements can affect the company. The description of the application mostly depicts the application from the view of an user and leaves the technical details very open.

Important points:

From the business case we will take what we believe are the most important points and highlight them to make it more clear what they mean.

Outsourcing:
It was clear that the project needs to make use of outsourcing to reduce the overall cost of the project. This can also be viewed as we need to have at least one developer active at all times in case of fault.

High availability:
Described as a “must have” in the scenario, the application must always be available to use. We must therefore ensure that we create a tech-stack and an architecture that has a high fault tolerance. As well we should also ensure a decent spread of servers and databases that ensures a stable latency no matter where in the world the user is.

Global:
The main purpose of the application is to promote Danish TV and films. This means that we need to stream video and audio to multiple users at a time without a break.

Features:
The user can see Danish films and TV programmes.
The user needs to be able to search for films and TV programmes through the use of search terms.
The user also needs to be able to give a rating to any film and TV program. It doesn’t specify if the user can rate each individual episode of a TV program.

User stories:

From the business case we will set up some different user stories that, while we will not be making an actual prototype, will help us get a better overview of what the application will need to be able to do and how.

- As a user I want to be able to create an user account on MyDRTV, so that I can login, rate shows and receive recommendations
  - Users can signup with email
  - Users can login with credentials
  - Users can edit their personal information on their account
  - The system must comply with gdpr

- As a user I want to be able to search for TV programs, so that I can quickly find content to watch
  - Users can search using keywords
  - Users can apply filters, eg. Genre, year of release.
  - Search results should be quick and the user should be able to select a movie/tv show from the result.

- As a user I want to be able to watch TV programs/Movies through the MyDRTV
  - The video should be able to stream videos smoothly
  - The user should be able to play, pause, rewind and fast forward videos
  - The system should be able track the videos track shows watched by the user to make a watch history for recommendations.
  - The system should have high availability so that users can watch without interference or interruptions

- As a user I want to be able to rate TV programs/Movies so that the ratings can be tracked
  - The user can give a score from 1-5 stars only once for a TV program/Movie.
  - The ratings should be aggregated so that the overall rating can be viewed by other users.

- As a user I want to receive TV program/Movie recommendations based on the my watch history and my TV programs/Movies ratings so that I can discover and watch new TV programs/Movies that interest me
  - The system should track watch history and ratings given by the user so that it can give recommendations based on them
  - The user receives recommendations based on only the MyDRTV catalogue,
  - The "More programs you may like" feature should show a list of content based on user preferences.

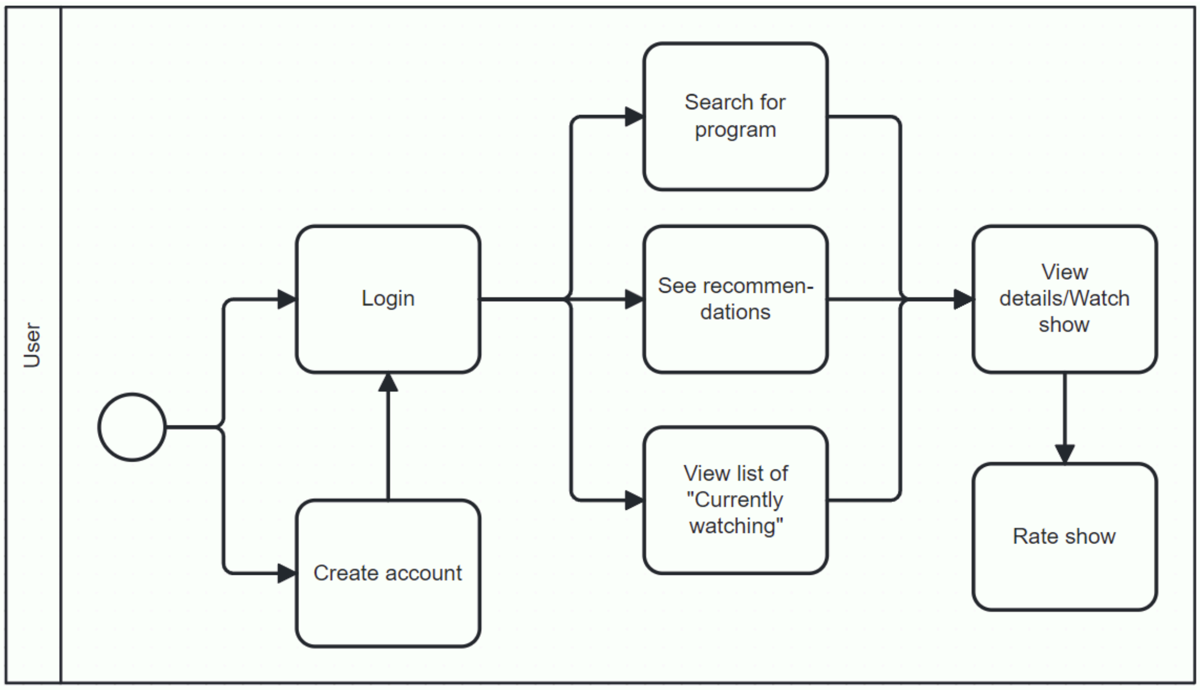
Brainstorm:

Here we will create diagrams and brainstorm how a prototype of the application could be constructed.

Diagrams:

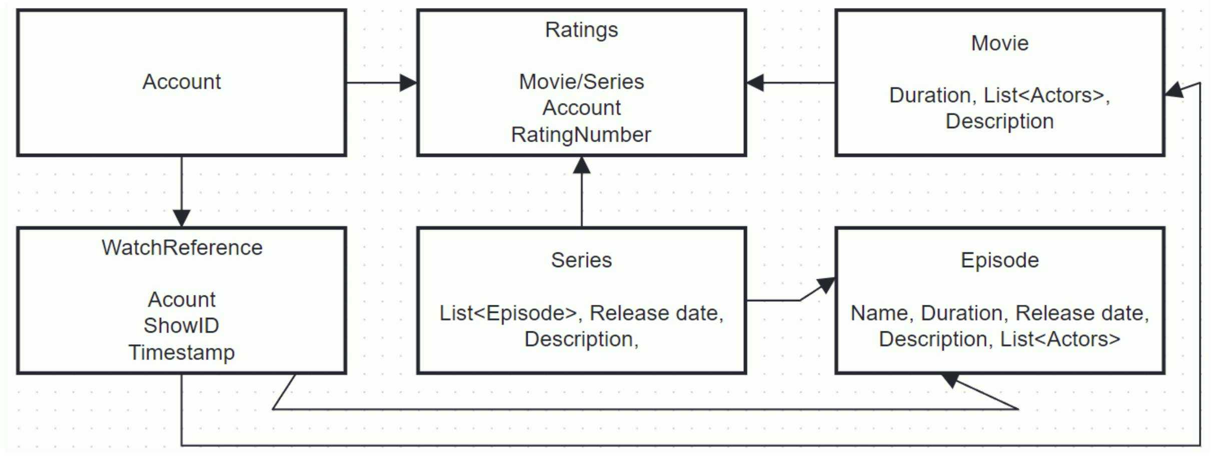
Here we will showcase and describe the diagrams we have created and why we choose these specific diagrams.

Flow diagram:



This flow diagram shows how a user would be able to navigate through our application. Initially the user has to log in. If the user doesn't have an account and is therefore unable to, an account needs to be created first. Once logged in, the user can find shows to watch by either searching for programs, looking through the recommendations or continuing the shows that are already in progress. Once having chosen a show, it can be either watched or rated.

Domain model:



This domain model shows the relations between the classes that would exist in our prototype application. Obvious variables like account email and name have been left out to prevent bloating the model.

Tech-Stack:

In this section we will select the technologies for the application. We have split the application into four different aspects, that being: Frontend, Backend, API and Developer tools.
For each of the aspects we have created a table where we have the name of the technologies, a brief description of what the technologie are, and lastly we have a brief description of why we have chosen that specific technology.

Frontend:

Frontend

Name:	Description:	Reason for use:
Typescript	Typescript expands upon javascript with the addition of types which make for easier development.	Javascript is considered a standard for website frontend development, so having the addition of types is why we chose typescript.
React (Redux)	React is a javascript/typescript framework for creating frontend functionality. It also contains the statemanager Redux.	We have chosen it for its popularity and because we are familiar with its usage.
Sass	Sass is a preprocessor scripting language for css.	Its use is intended for making development in css easier to look good compared to working with basic css.
TailWindCSS	Tailwind provides pre-built css classes.	Due to css often being difficult to use when creating new classes, we instead found a framework that has many classes already built.
PostCss	Post is a framework for adding functionally to css.	Again we have a framework for making css development easier.

API:

API

Name:	Description:	Reason for use:
GraphQL	GraphQL is a query language for building up an api.	Making scaling easier.
Apollo	Apollo makes GraphQL work for you at any stage and any scale.	Working with GraphQL to improve its scaling capability while also providing tools for improvements.
Stripe	Stripe handles payment of real world currency.	During research, we found Stripe to be a very popular payment for handling microservices that generally has a great reputation.

Auth0	Auth0 is an easy to implement, adaptable authentication and authorization platform.	We need a way for the user to securely log in to be able to properly handle orders and Auth0 appears to be able to handle everything we need while also being popular.

Backend:

Backend

Name:	Description:	Reason for use:
NestJS	NestJS is a Node.js framework for building efficient and scalable server side applications.	We have chosen to use NestJs as it provides a lot of structure through a modular architecture. It also has good integration with both Typescript and TypeORM.
Node.js	Node.js is a Javascript runtime environment that enables running Javascript on the server side.	Using Node.js allows us to use Typescript for both frontend and backend, which simplifies development. Node.js also has a wide range of libraries and frameworks available through the node package manager (npm).
Nginx	Nginx provides a web server, and can also serve as a reverse proxy and HTTP cache.	We use Nginx as a reverse proxy and for load balancing to route requests to different backend servers, which improves availability and scalability.
TypeORM	TypeORM is an ORM framework for JavaScript and Typescript, which uses object-oriented programming principles to help manage database operations.	TypeORM allows us to define our database schema using Typescript classes, which helps work with relational databases in an object-oriented way.
Docker	Docker allows packaging applications into containers, providing consistency across different environments and making it easy to deploy	Docker ensures every team member has the same environment, removing the risk of version based errors and configuration inconsistencies
Digital Ocean	DigitalOcean is a cloud based platform that provides virtual servers and services for deploying/scaling applications.	We have chosen to use DigitalOcean since it allows us to easily deploy our application.
Neo4j Graph Database	The Neo4j based graph database is a nosql database wherein we don't create rigid relationships between tables.	We have chosen to use a graph based database because of its loosely based relationships and for its scalability.
Internal micro -services	Considering we would be making a website for a business that most likely would include other departments like the bakery, accounting, storage and so on, they would most likely have some service set up to handle each area which we wouldn't be making/developing.	Any department within the business which has their own systems set up to handle their part of the business, we would connect to them via an api they have set up. This also separates the responsibility to each department and makes the development of the web store easier to handle.

Developer tools:

Development Tools

Name:	Description:	Reason for use:
GitHub	Github is a developer platform that stores and manages code using Git to provide the distributed version control. It features continuous integration with test authentication to ensure no failed build will be pushed to the user.	GitHub is one of the most popular version control tools used in the field. It is also the one we are most familiar with and adding onto that it also has continuous integration built into it with tests we can make ourselves.
Visual Studio Code	Code IDE capable of being used with multiple different programming languages.	We are taught in using this IDE as part of our education, so we know it is capable.
Jetbrains Intellij	Code IDE capable of being used with multiple different programming languages.	We are taught in using this IDE as part of our education, so we know it is capable.
Trello	Project management tool that helps organize tasks using boards, lists and cards	We were introduced to Trello though our education and has since benefitted from the overview and structure it provides
Webpack	A static module bundler. When webpack processes your application, it internally builds a dependency graph from one or more entry points and then combines every module your project needs into one or more bundles, which are static assets to serve your content from.	Since we, in our frontend of our technology stack, use multiple technologies for ease of development in css and typescript, we found a bundler we know will work with them all.

Architecture:

In this section we will go over the different criteria that we set up based on our tech-stack and our findings in the business case section and the brainstorm section of this report.

Application criteria:

Here we set up different criterias based on the work assignment document. We have the technical aspects: Agility, Simplicity, Scalability, Fault Tolerance, Performance, Extensibility, and then we have the economic aspect: Overall cost.
We will set up each of these aspects and describe their individual importance to the project and through that we can make comparisons to architecture patterns and select the one we believe best matches our criteria.

Agility (5/5):
We have set agility quite high at a 5/5 since the final product being developed is a streaming service and from our understanding there is a lot of continued work needed to ensure performance.

Simplicity (3/5):
We've placed the simplicity at 3/5, which means rather simple. This is based on the fact that an account creation/log in system is required for most applications. The difficulty is introduced based on the backend. Handling the shows and streaming of content is what increases the difficulty. This is partly because our development team is rather inexperienced when handling streaming and movie/series storing, plus it being a somewhat uncommon task.

Scalability (3/5):
The scalability has been set based on the expectation that movies and series, as well as individual users, will continue to grow as the application becomes potentially more popular around the world.

Fault tolerance (5/5):
We have placed fault tolerance at a 5/5 because the business scenario specifically made it a point to mention that the firm's reputation is at risk because it has been placed on achieving a widely used application.

Performance (4/5):
Performance has been set at 4/5 because of the use of a vide network of databases and servers. Having multiple databases to store shows and servers to connect to, almost guarantees there is data available and easily accessible. Spreading users and connections prevents/reduces wait time when encountering high amounts of concurrent users.

Extensibility (3/5):
The overall assignment doesn't include a lot of functionality, so we've chosen to increase the extensibility since adding new features doesn't impact the system on a large scale.

Overall Cost (4/5):
While the description and scenario of MyDRTV indicates that they want to reduce overall cost of the development process, they also indicate that the application will need to make use of multiple servers located around the world which will increase the cost. Because of this, we have given the overall cost a 4/5, but that doesn't mean that our selection must meet this cost.

Comparisons:

To ensure high performance, scalability and fault tolerance, we have chosen to store the shows on multiple databases found at different locations. Even though this increases scalability, and performance, it also greatly increases the overall cost, since database storage in high amounts is a costly decision. Our approach is highly agile, which is a choice we've made since it provides the best user experience with continuous updates and improvements. This does increase the amount of structure/team coordination needed, but is considered worth in the grand scheme. To ensure the best possible user experience is also an important contributing factor when deciding to split out data across multiple databases. The purpose of the application is to watch shows and when a user has to wait for the only purpose of the website, their satisfaction is substantially reduced.

Event Driven

We have chosen event driven architecture for the MyDRTV platform. This decision is based on its scalability, fault tolerance and performance.
The system needs to be able to scale globally, supporting potentially millions of concurrent streaming users. In an event-driven architecture, components can scale independently. If one of the services spikes in users, for example search recommendations or streaming, the related services can be scaled and optimized for usage. Another benefit with the event-driven model, is the support of horizontal scaling.
Event-driven systems naturally support fault tolerance. If one service fails, for example the recommendation service, the other services should be able to run without failure, as events are queued and processed once the system is up and running again. This aligns with our requirement for high availability.
Event-driven Architecture achieves high performance, because services process events asynchronously. Asynchronous processing allows the system to handle large volumes of requests without slowing down the application. Event based architecture allows services to be triggered by specific triggers, improving the responsiveness of features.Overall it reduces bottlenecks which is an important feature for the system.