Graduation project report

Interactive overlay controlled with serverless technologies

December 2022

Student.
Magnus Stuart Juul
Majuu19@student.sdu.dk

Supervisor.
Mina Alipour
mial@mmmi.sdu.dk

Abstract

Preface

This report is written to document a solution to a 7th semester software technology graduation project, the report concerns a system which contains an overlay capable of anchoring itself on top of a video on a website, and then get controlled using AWS serverless technologies, which can be operated from a control center. The report aims to document the process, key decisions and other thoughts which led to the prototype.

Acknowledgements

I would like to acknowledge the following

Mina Alipour

Mina has been the supervisor of the project, she has provided great feedback and help, whenever it has been needed.

TV 2 Fyn

The project has been developed in cooperation with TV 2 Fyn, and I thank them for the opportunity to do so.

Kasper Skov Hansen

Kasper is an employee at TV 2 Fyn, he has been a great sparring partner for discussing design questions regarding the software architecture, and other code related questions.

Louise Koustrup

Louise is an employee at TV 2 Fyn, she has been a great help for decisions on design questions regarding the look of the overlay.

Reading guide

The project is structured in the same order as certain thoughts and decisions were made, alongside with when different parts of the system was developed, therefore it is recommended to read it chronologically. However, if the reader wishes to get a fast overview of the report, alongside with the key ideas, it is recommended to first read the Abstract followed by the conclusion. The target group of the report is software technology/engineering students which have at least completed their 7th semester.

Table of contents

Introduction

Background

With the increasing digitalization of our daily life, which has contributed to the switch from traditional TV, that being both Analog and Digital TV signals, to a more web and streaming based consumption. This has naturally led *TV 2 Fyn* to look more at how they can adapt their platform and product(s), to accommodate this switch, and thereby deliver a product that their viewers, the Danish population, primarily, on Funen. Inspired by platforms such as *Twitch.tv* and *Youtube.com*, which offers a place for individuals to stream everything from video games, music practice, everyday life etc. all while giving the streamer the possibility to interact with their viewers.
This last part, the interaction, is what is thought to make the difference between a small unknown streamer, and a big known streamer, and studies has found that interaction increases so called *SOVC (Sense Of Virtual Community)*, that will then lead to an increase in viewer retention.
And that is what this project has aimed to try and solve, by giving *TV 2 Fyn* the possibility to interact with their viewers from home, and thereby increase their *SOVC*.

Problem definition

The following problem definition and its corresponding sub-questions, which will get answered throughout the report is as follows:

"How can TV 2 Fyn increase interaction with their viewers, and thereby create an SOVC?"

- "Does some functionality on the TV 2 Fyn platform already exist, which may be utilized for this?"

Later when the idea of the interactive overlay came up, these sub-questions arose.

- "Does it make sense to look at serverless technologies for this?"
- "Is it viable for TV 2 Fyn to implement the changes in broadcasting, that this may entail?"
- "Does TV 2 Fyn have any shows or broadcasts, where the interactive overlay could fit in?"

Project boundary

In the project, the focus has lied on ease of use for the end-user/TV 2 Fyn viewer, alongside with a focus on making the implemented overlay options robust and stable, instead of developing a plethora of options. Therefore the overlay only supports two kinds of options, but it will not be difficult to extend this list; the next focus lied on the implementation of AWS, and thereby making the logic for the overlay serverless; there has not been focus on making the overlay easy to use for a TV 2 Fyn employee, and the design and content of the control center has not been in focus.

Purpose and goal of the project

The aim of the project was to create an overlay, which could be anchored to a video on a webpage, where the overlay would then sit inside the video container. Also, to redirect load from the TV 2 Fyn servers the triggering of the overlay and storage of answers was made serverless, by using the different services offered on AWS, especially the services: AWS Lambda, Amazon DynamoDB and Amazon API Gateway.

The project is relevant, as it might show the skills and capabilities of the student(s), skills and capabilities that should have been developed during the 5 semesters, with the courses and classes and lastly as a final test after the internship that took place during the 6th semester.
On top of that, what the project has produced is a system that could be used by TV 2 Fyn, to try and adapt to an everchanging world, and maybe even thrive, by being frontrunners by adapting technologies not yet seen in the world of news.

Technical knowledge base

Notations and terms

Technical term	Description
AWS	Amazon Web Services. A provider of cloud computing services and API's.
Serverless	Also known as: IaaS (Infrastructure as a Service)
AWS Lambda	An event-driven, serverless computing platform service. Offered by AWS.
Amazon DynamoDB	A fully managed proprietary NoSQL database service, that supports key-value and document data structures
Amazon API Gateway	A service that makes it easy for developers to create, manage, monitor, and secure APIs at any scale. And easier control the flow/access to systems and programs.
Overlay	A collection of overlay contents (see below), usually a certain broader topic. An overlay can contain multiple overlay contents.
Overlay content	Represents the different questions/polls contained in an overlay. Overlay content can only belong to a single overlay.
Control center	A GUI developed in VueJS, that allows non-tech savvy employees of TV 2 Fyn to monitor and activate the interactive overlays.
Gamification	The concept of integrating similar experiences to those experienced in games into other activities, by using game design elements to improve user engagement.

Table 1: Notations and terms used in report.

Prerequisite knowledge

The project is based on knowledge from previous semesters, alongside with knowledge gathered by different TV 2 Fyn employees, through their professional career. The chosen programming language for the project is JavaScript/TypeScript, therefore it is recommended that the reader understands these/this language, but it is not a requirement.

Similar tools review

Brightcove

Brightcove[1] is a Massachusetts-based software company that offers an online video platform, this online platform includes their own player, which is built on top of the hugely popular open source HTML5 video player video.js.
Brightcove is used by a variety of big and known companies, such as, *Adobe*, *BBC*, *Ford*, *MasterClass*, *The Premier League* etc. as *Brightcove* offers almost everything inside the spectrum related to online video streaming.

The overlay offered by Brightcove

Brightcove does offer an overlay for their video platform/player as part of their "Brightcove Interactivity"[2], the display of the overlay and its different parts; according to *Brightcove's* own documentation[3], it can be controlled by player events such as pause, play or other custom events, or time intervals which allows for displaying an overlay during a certain period in the video.
There is then the possibility to add images, text, HTML, or a DOM element, to the content of the overlay, alongside with controlling where the overlay should be placed in the video; it is also possible style the overlay to your own liking with custom CSS.

This makes the solution offered by *Brightcove* ideal for pre-produced videos and live streams, where there is not a need for fine control of when overlays get triggered, such as having a watermark leading to a certain page or a banner with information regarding the live stream.
This makes the product(s) offered by *Brightcove* optimal for companies looking for a video platform, capable of hosting their pre-produced video-content.

What is lacking?

The overlay functionality offered by *Brightcove* will most likely serve most of the needs and wishes from a large majority of their customers, however, in the world of live streaming, one functionality is missed, the possibility to accurately control the triggering of overlays, this is a major thing for news stations such as *TV 2 Fyn*, as a live broadcast almost never sticks completely to the planned schedule, therefore time coding the triggering of events is not a viable solution, as it may result in the triggering of overlay content before it is relevant, or in some cases, even the triggering of overlay content, that may not be appropriate due to certain conditions/events.
An example could be when Christian Eriksen collapsed due to a Heart Attack during the Euro 2020 match between Denmark and Finland, in theory an overlay asking: "Who do you think will win the match?" could have been set to trigger somewhere during that match, and this could have caused a major outrage if that had happened.

Lastly, the fact that *Brightcove* offers the overlay as an extra service for their video platform, could also be a deal breaker for some, as not all might be interested in switching away from their current video platform, might that be a self-hosted or a competitor to *Brightcove*, and nothing seems like that it is possible to solely subscribe to the overlay functionality.

[1] [Brightcove]: https://www.brightcove.com/en/
[2] [Brightcove Interactivity]: https://www.brightcove.com/en/products/interactivity/
[3] [Brightcove overlay documentation]: https://player.support.brightcove.com/plugins/display-overlay-plugin.html

Vixyvideo

Vixyvideo[4] is a software company based in Hilversum, Netherlands, focusing on video platform software, such as their product "Interactive Video"[5] which is an add-on to their product "Online Video Platform"[6], which is a video hosting service for your videos with their video player built on top of the Kaltura[7] video platform.

The overlay offered by Vixyvideo

Customers on the lookout for a video platform, of which they use for advertisement of their products on their own page, will most likely see *Vixyvideo* as a strong and interesting option; this stems from the reason that *Vixyvideo*'s Interactive Video product, clearly directs itself towards that market, by using advertisement as its primary use case, however, the more interesting one in this case, is their idea to use it for "Engagement", to help turn passive viewers into engaged viewers.
The Interactive Video can then contain different types of overlays, such as buttons, action areas, text etc. which all may have their own associated action(s), such as opening an URL while jumping to a timecode followed by pausing of the video.

This is ideal for companies looking for a solution for their product page, as it seems that this has been their focus, however, as they mention on their page it can also be used for customer support, e-learning, engagement and more. By offering the possibilities to tailor the video behavior to the selections by a user, thereby offering multiple storylines to unfold, or giving different answers to customers depending on the questions answered throughout the video.

What is lacking?

The Interactive Video product offered in conjunction with the Online Video Platform, does seem like it will fit like a glove for customers looking for an innovative way of displaying and advertising their product catalog, and it would probably be a good fit for customers with a lack of tech-savvy employees, as it seems like an easy batteries-included no-code solution. This, however, does have its own drawbacks, as you are more constrained in your creative freedom since there is not the possibility to add custom HTML or DOM elements.

The biggest drawback from the perspective of this project is the same as *Brightcove*, no support for controlling the overlay other than time coding, which is a crucial thing for customers broadcasting live, as a live transmission never sticks 100% to the schedule, and a live transmission needs to have the capability of reacting to conditions and events that may happen during the live broadcast.

And lastly, a negative thing in the eyes of some, and a positive thing for some, the need to buy and use the Online Video Platform offered by *Vixyvideo* to use the Interactive Video, may be a deal breaker for some, as they might not be interested in switching away from their current video hosting solution, whatever that may be, which means it could be quite intrusive to implement on your website(s).

[4] [Vixyvideo]: https://www.vixyvideo.com/
[5] [Vixyvideo – Interactive Video]: https://www.vixyvideo.com/interactive-video/
[6] [Vixyvideo – Online Video Platform]: https://www.vixyvideo.com/video-platform/
[7] [Kaltura]: https://corp.kaltura.com/

DOT.VU

DOT.VU[8] is an interactive video content platform based in Holstebro, Denmark, with the mission to make interactive content creation available for everyone, without any need for coding experience, and thereby enabling personalized interactive content. In the view of this project, their product "Interactive Video"[9] is the most interesting one as it offers: shoppable videos, branching videos, and personalized videos, while following their mantra of no coding required.

The overlay offered by DOT.VU

Companies with online product catalogs and stores, will most likely see the idea of *DOT.VU*'s "shoppable video", as a very interesting feature to implement on their site, as this could help to increase sales, using gamification or similar ideas.
Another use case could be for companies wishing to offer videos with different outcomes, be that a product videos, where the user gets to experience a variety of choices in the production of a product, which results in different products in the end, or be it a TV-series, where the user gets to decide what the main character does.
Lastly, it could also be used to create personalized videos for users, such as offering a service where a user can select from a catalog of videos with celebrities, each with their own settings, in where they can create personalized videos for birthdays, travel etc.

This makes *DOT.VU* an interesting option for companies or sites, looking for a complete video package, where in no coding experience or know-how is required, all while offering interactive options to be added onto the videos.

What is lacking?

As it often is with solutions offering a no code approach for the consumer, creative freedom can quickly suffer, as the only counter-measure to this is that the developers of *DOT.VU* would need to be the first ones to implement the feature, and then make it available for their customers, or be ready to develop and implement if the customers come to them with an idea, both of which probably are pretty unlikely.
DOT.VU might also lose the interest of customers not looking to replace their current video platform, that might just be on the lookout for an overlay they can implement with their already existing video platform.

Lastly, nothing points toward the possibility to control the triggering of the overlay on multiple user's machines simultaneously, making it lackluster in terms of being used for live streamed events.

[8] [DOT.VU]: https://dot.vu/
[9] [DOT.VU – Interactive Video]: https://dot.vu/interactive-experiences/interactive-video

Bridging the gaps

After reviewing the 3 tools offered by *Brightcove*, *Vixyvideo* and *DOT.VU*; nothing really exists to fill the gap in the world of live video broadcasting, which probably is the reason why interactivity in broadcasts by media houses, have not been seen yet, this is the gap that the overlay described in this report aims to bridge.

First, by offering a solution where the benefits from interactivity can be used in a viable and controlled way with live broadcasts, by utilizing the benefits of serverless architecture to orchestrate the triggering overlays; topped by the benefit, of serverless architecture, for smaller companies, that may not have the economy or infrastructure to host and maintain their own servers.

Second, with it being independent of and anonymous to the video player, and thereby giving the freedom to select whichever video platform a company desires, which for some can be seen as a benefit, and for some seen as a negative, but the nature of the overlay would also allow any of the video platforms reviewed, to implement the overlay with their existing solution, and thereby increase the amount of services offered by them.

Method

Unified Process

This project is structured using the Unified Process (UP), by some known as Rational Unified Process (RUP), which is an iterative software development process, as this process have yielded great results in projects during former semesters; the project makes use of the first 3 phases: inception, elaboration, and construction, and will not enter the fourth phase of UP, the transition phase, as this would be out of scope, and not within the timeframe for a graduation project.
The inception phase in a project using UP, focuses on establishing the overall requirements for the project, this is followed up by work to identify the critical risks of a project, and how to mitigate said risks.
Then the elaboration phase, where the focus lies on delving more into the details of the project, by constructing a more comprehensive list of requirements, from the overall requirements in the inception phase, and modelling these requirements into the design of the system.
The construction phase, the system is build using the requirements and models from the elaboration phase as the guide, and compendium for decisions.
Lastly, the transition phase, concerns itself with deploying the system for the client, alongside with adjusting the system based on feedback from the client.

The inception phase, started off with the need for a starting point for the project which amounted to the writing of the Project description, where the purpose of the project description was to describe the envisioned project, by describing the scope, problem, solution and technology, alongside with looking for similar tools, and taking them into consideration, and why this project would fill a gap, that they did not already fill (see Similar tools review for more).
Lastly an attempt to identify the critical risks was performed, to help plan for mitigation of these risks, should they arise.

Following the inception phase, the elaboration phase, a more complete list of requirements was constructed taking the basis of the preliminary requirements from the project description, these requirements were then run through an MoSCoW-analysis, to help prioritize them, of which the most complex and critical ones has been modelled. Then the system was modelled, and different UML-diagrams was used, for some of the more complex and critical operations, all while laying out the plan for the software architecture. This was followed up by the construction of a conceptual prototype, to test if the envisioned solution is possible, and if any changes would have to be made, to accommodate for potential shortcomings in the envisioned software architecture.

Lastly, in the construction phase, the real construction of the system could begin based on the models and designed software architecture from the elaboration phase, as this project was not performed in a team, the reason to use Scrum did not seem to exist, as it is a team discipline.

Agile

As mentioned in the former section, the project never entered the transition phase, but as a project always has the benefit of at least some amount of feedback, some of the Agile concepts was implemented, such as the need for frequently recurring feedback from the client, in this case *TV 2 Fyn*, this was done by weekly and sometimes daily small standing meetings by my desk, where the current progress was shown, followed by no holds barred feedback from *TV 2 Fyn* employees, such as how the overlay looks, how it is used and more.

This, while not being the textbook way to do it, resulted in the development of a system, that already had some of the wishes from client already implemented into it, or at least thought into it, so that it would not be a big hassle to implement these features into the system later.

Project Organization

To keep track of the progress of the development different requirements, the tool *Asana*[10] was used, as it offers a variety of views for representing progress of requirements/goals in a project, this also had the benefit of being a tool already used at *TV 2 Fyn*, which made it possible for employees to follow the progress, all while fitting into their daily routines of using *Asana*.

[10] [Asana]: https://asana.com/

Requirements

Functional requirements

ID	Name	Description	MoSCoW
F1	GUI to control overlay (control center)	A TV 2 Fyn employee (or similar), must be able to control and monitor the overlay from a GUI	Must
F1.1	Relevant overlay content triggers	The system must be able to trigger the overlay content only of an overlay on relevant users and not on irrelevant users, such as which users have already given their response for this overlay content.	Must
F1.2	Monitor overlay responses	The system must be able to monitor and present the overlay content responses.	Must
F2	Overlay	The system must have an overlay which can contain a variety of overlay content.	Must
F2.1	Automatic overlay anchoring	The overlay must be able to anchor itself to any video player on a website, such as on the tv2fyn.dk website.	Must
F3	Overlay management	The system must be able to save data about overlays, and its’ overlay content and user responses, in a NoSQL Database.	Must
F3.1	Overlay creation	The system must be able to create new overlays containing overlay content.	Must
F3.2	Respond to overlay content	The system must be able to receive responses to overlay content	Must
F3.3	Get relevant overlay	The system must be able to send the relevant overlay and its’ overlay content to a user.	Must
F4	User management	The system must be able to save relevant information about users, in a NoSQL Database.	Must
F5	User distinguishing	The system must be able distinguish between users.	Must

Table 2: Functional requirements for the system.

Non-functional requirements

ID	Name	Description	MoSCoW
NF1	Usability	The overlay must be responsive, and intuitive for the user, thereby resulting in a smooth experience.	Must
NF2	Performance	When a TV 2 Fyn employee (or similar) triggers overlay content of an overlay on users, it must reach the users within 2 seconds.	Must

Table 3: Non-functional requirements for the system.

Detailed requirements

Use case table

Based on the functional requirements, use cases were created containing the various requirement(s), needed for the use case to be carried out, the outermost right column shows which functional requirements each use case contains/depends on (see Table 4).

ID	**Name**	**Description**	Contains
U01	Trigger overlay content	A TV 2 Fyn employee, must be able to select the overlay, followed by what overlay content, that they wish to trigger on the relevant users.	F1.1, F4
U02	Monitor overlay responses	A TV 2 Fyn employee, must be able to select the overlay that they wish to monitor, and then get a representation of responses for the overlay content(s) on the overlay.	F1.2
U03	Automatic overlay initialization	When a user connects to a page with a video player and overlay, the overlay must automatically anchor itself to the video player, followed by requesting the AWS backend for the relevant overlay and its' overlay content.	F2.1, F3.3, F3, F4, F5
U04	Send overlay content response	When a user has selected their response, they must be able to send it to the AWS backend, where it will get saved, and the system updates information about responses for the overlay content, and that the user, have given their response to this overlay content.	F3, F3.2, F4, F5

Table 4: Use cases for the system.

Use case diagram

To illustrate the combined functionality of the use cases, provided to the users of the system, a use case diagram was created.
The diagram depicts the relationship between individual use cases and primary actors, "TV 2 Fyn employee" and "User", alongside with secondary actors, "User database", "Overlay database" and "Relevant overlay users", which is a general depiction of users who are eligible for triggering of overlay content.

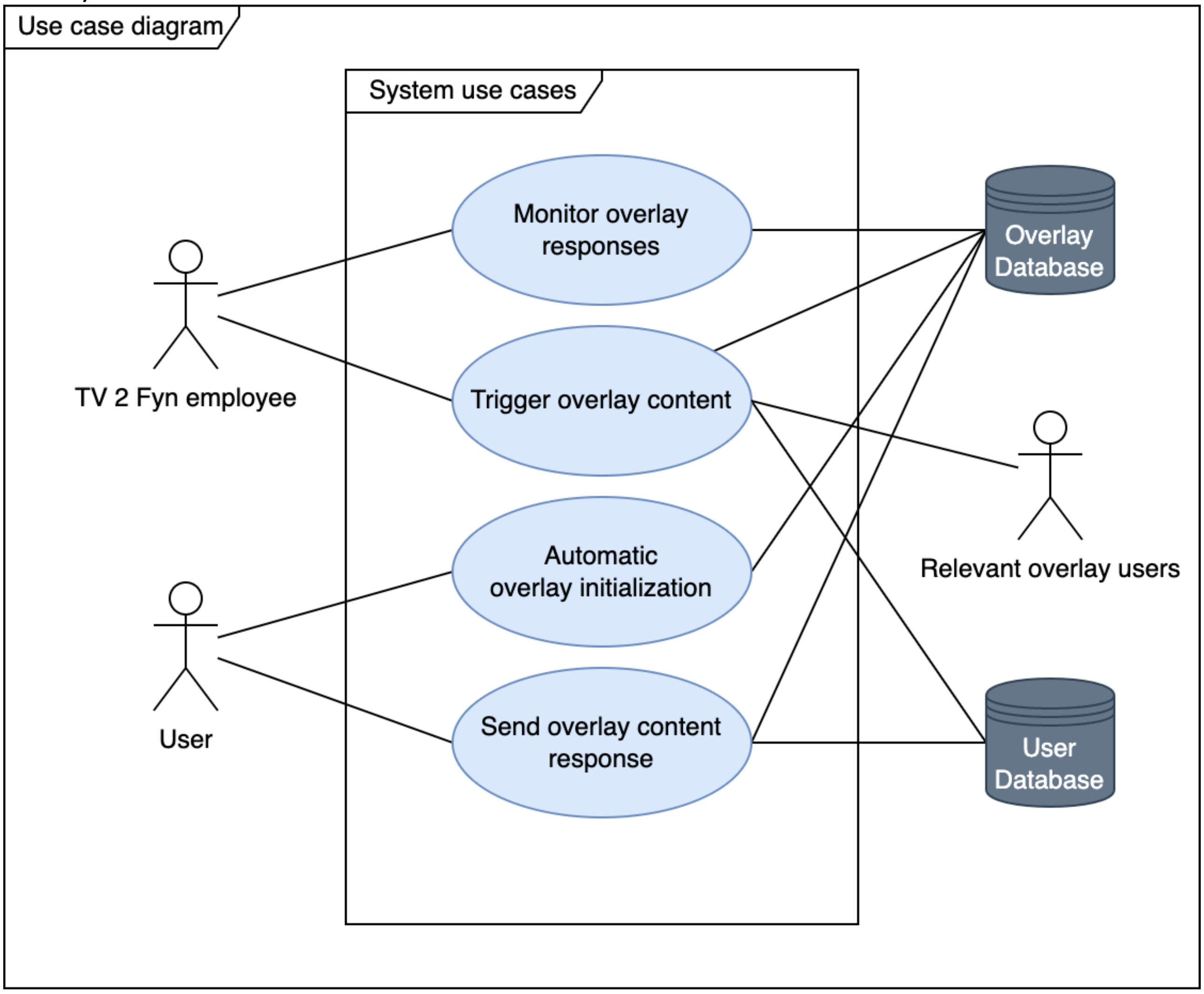


Figure 1: Use case diagram for the system.

Design

Software architecture

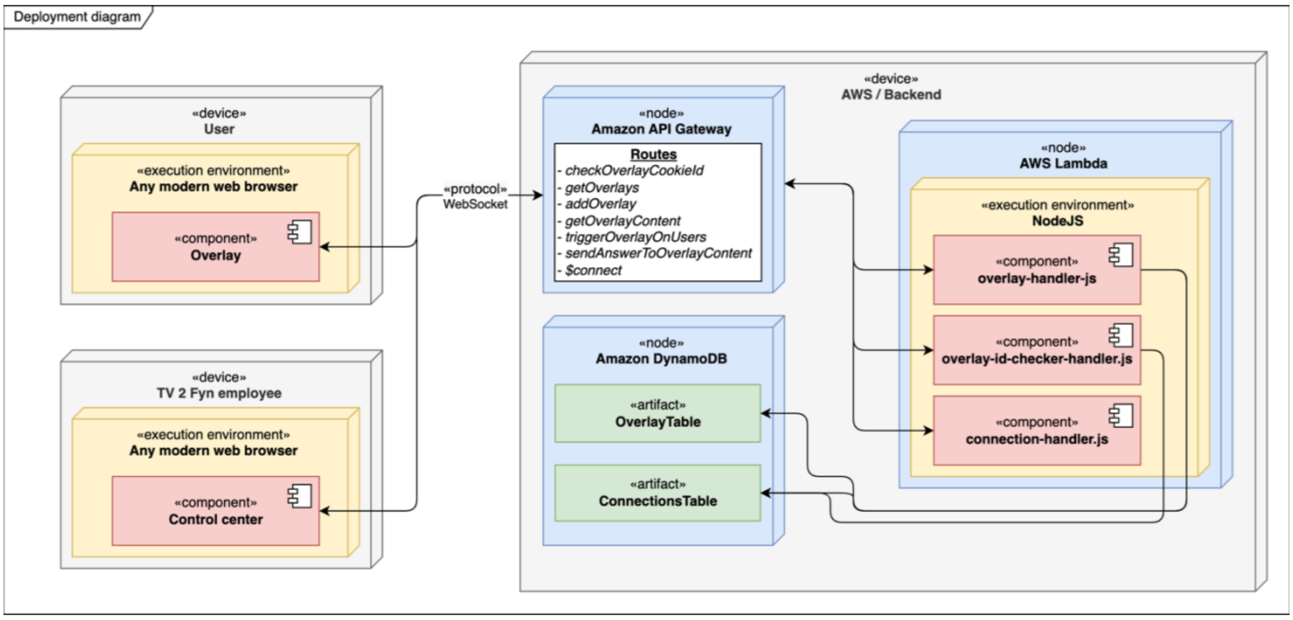


Figure 2: Deployment diagram of system

The deployment diagram (see Figure 2) represents how the software architecture has been envisioned and thereby how it has been designed, alongside with which of the different parts of the system, that interacts with each other. The colors selected for the deployment diagram, has no meaning, and have solely been selected so that it is easier to distinguish between parts.

The deployment diagram is designed as if the *Overlay* and *Control center* runs on two different machines, this however, does not mean that this is a requirement, and the system has been tested with three separate computers running, where two ran the *Overlay*, and one ran the *Control center*.

To start with the *User* connects to a page containing the *Overlay*, then the *Overlay* for the user connects to the Amazon API Gateway, by the use of the *$connect* route with a WebSocket connection, enabling a two-way communication, after the user has connected, the overlay sends the value of the cookie used to identify the user, through the Amazon API Gateway with the *checkOverlayCookieId* route, this invokes the *overlay-id-checker-handler.js* component, which sends back an appropriate response to the user depending on whether the user exists in the database. When the overlay has received a response, the *getOverlayContent* route is requested, which in turn invokes the *overlay-handler.js* component, that either sends back an error response or the overlay content for the requested overlay, depending on the body of the request from the user. Following the response from the *getOverlayContent* request, the *Overlay* loads in the response data, and the *Overlay* is then ready to get triggered.

For the triggering of the *Overlay* to happen, a *TV 2 Fyn employee* needs to access the *Control center*, when the *Control center* site has been loaded, it connects to the Amazon API Gateway, just like the *Overlay* with the "$connect" route, following a successful connection, the *Control center* sends a request to the *getOverlays* route which invokes the *overlay-handler.js,* that retrieves the overlays from the *OverlayTable*.

Following this, all overlays, and their contained overlay content, this is then loaded into the GUI of the *Control center* site, and it becomes possible to select an overlay, which will load the responses for each overlay content, and lastly it becomes possible to select the overlay content of which the *TV 2 Fyn employee* wishes to trigger on the relevant users. When the overlay content is desired triggered from the *Control center*, the *triggerOverlayOnUsers* route is requested, invoking the *overlay-handler.js*, that based on values in the body of the request, lookups relevant users in the *ConnectionsTable*, gets their unique connectionId, which is used to send the Id of the desired overlay content to all relevant *Users*, where the *Overlay* is waiting, when the *Overlay* receives this Id, the desired overlay content is shown to the *User.*

When the *User* then sends back their response, the *sendAnswerToOverlayContent* route is requested, which in turn relays this on to the *overlay-handler.js* that depending on the type of overlay content, adds the responses to the corresponding overlay content in the *OverlayTable*, followed by registering that the *User* has responded to this overlay content in the *ConnectionsTable*.

Overlay

Frontend

The Overlay has been designed to be intuitive and easy for users, without any introduction on how to use it, while being lightweight for the browser to run, so that it does not degrade the experience of other parts of the page, especially the video player. Another aspect that was thought into the design, is the possibility to develop custom components to Overlay content, for the Overlay to load and show, as the software design is meant to be as developer friendly as possible.

With the combination of existing usage of Vue.js for the tv2fyn.dk website, and the experience in using Vue.js for the student, made it the ideal choice for achieving the above, as it offers the possibility to develop custom components and load them into the Overlay component, with only a few demands for the software design of the custom component(s), while being a modern and performant JavaScript framework with TypeScript support; It is however worth to mention that TV 2 Fyn still uses Vue 2, while the Overlay is written in Vue 3, but as TV 2 Fyn is during a migration to Vue 3 at the moment, it was decided to be the most future proof decision.

The choice of using TailwindCSS, was also made as the tv2fyn.dk website, is in the process of having all its styling rewritten in TailwindCSS, and the student saw it as an excellent opportunity to start exploring the CSS tool, while making the Overlay compatible with the future of TV 2 Fyn.

The design of the Overlay has been made with a focus on simplicity, hoping it will be easier and more manageable for new users of the Overlay, on top of this, the design has also tried to be as little intrusive to the viewing experience as possible, meaning that it will not take up the entire video player, but instead constrain itself to the bottom left corner, with just the space needed for the overlay content elements, with the title at the top, a list of options in the middle, and the send response button in the bottom, alongside with a small padding, making it seem less cramped.
One idea that was floated were also the idea of whether it should be possible to hide and show the Overlay on demand, which might have been a nice feature for some users, but for some it could have been an unnecessary complex feature.

As of right now, the user needs to respond to the Overlay, before it will get hidden again, and if a user is to slow to respond, while another piece of overlay content is triggered, the former overlay content will simply get replaced, and the user will no longer be able to respond to it, this however, will not register as if the user has responded to the former one, and if it is triggered again later in the broadcast, it will once again appear. It also worth to mention that the Overlay will work on browsers on mobile, but it has not yet been optimized for it, therefore some dimensions might be off.

Control center

Frontend

While the Control center has not been a center of attention for the project, with most of the attention being directed towards the Overlay and Serverless part, the design is still worthy of a short mention.

Like the Overlay, the Control center has been made using the Vue.js framework, and more specifically Vue 3, with the use of TailwindCSS for the styling. The decisions on why, are the same as the ones for the Overlay, and it seemed natural to develop the parts of the system with the same tools, making it more uniform, and easier for future developers to work with.

The Control center has been designed to give a fast and easy overview of options and results, this is done by offering an almost blank page, with a dropdown menu, able to select one of the automatically retrieved overlays, when an overlay is selected, the overlay content of the overlay will be able for selection in another dropdown menu, and the current response statistics for each overlay content will be visible beneath, with a clear title and options show in a list.
Following the selection of an overlay content, clicking the button with the text: “Activate overlay content on users”, will trigger the overlay content on all relevant users; following triggering of overlay content, it may be expected that some answers may start to come in, and instead of flooding the backend with requests, it is possible to click 2 buttons: “Refresh data” or “Refresh data every 10 seconds”, where the latter will refresh the values automatically every 10 seconds, or until deactivated.

The Control center has not been built with mobile in mind, as it both was not a piece of much attention during the project, and since it is expected that controlling it from mobile would not be its primary platform.

Serverless

Backend

The backend has been created using JavaScript, as it is the same language used in the frontend for both the Overlay and Control center, thereby making it easier for future developers to maintain and develop new features to all parts of the system.
This decision is also supported due to Node.js, which is a backend JavaScript runtime environment, being one of the natively supported environments for AWS Lambda, thereby making development easier and faster.

Some parts of the backend however are not JavaScript per se, Amazon DynamoDB (which will get addressed in the next sub-section (see Persistence), and Amazon API Gateway, which is both defined, named, connected and setup from the template.yaml file in the root of the folder for the serverless code, this makes the code easily deployable on all AWS accounts.
As the AWS Lambda, which is the logic part of the backend, is not accessible directly through a domain name, this must be done through the Amazon API Gateway, with each route key acting as a proxy, to the corresponding Lambda file, as an example, the *$connect* route acts a proxy to connection-handler.js, and the *getOverlays* route acts a proxy to overlay-handler.js. This is done as it enables the possibility to keep the same domain name, but have different stages (versions), where new functionality or behaviors can get tested without breaking existing implementations.

When a user of the Overlay or Control center, connects to either of the sites, a WebSocket connection will get created between the backend and the users web browser accessing the site, from here the different routes can get requested by sending a message to the backend, the message is a JSON object, that has to contain an "action" key of which the value decides what route to request, an example of such a value could be: *getOverlays*, to pass additional data that the corresponding logic to each route contains, often another key has to exist inside the message object, and that is the "content" key, which does not contain a single value, but an object with multiple key-value pairs, of which differs depending on what route is being requested.
Two examples of such messages can be seen below:
For the *sendAnswerToOverlayContent* route:

```
{
  "action": "sendAnswerToOverlayContent",
  "content": {
    "overlayId": "c4fa173a-8040-4410-b338-079771064f29",
    "overlayContentId": "526cd653-fc7d-4c53-90a3-9c04ef78097e",
    "answer": "some_example_answer",
    "overlayCookieId": "e3949c5a-3d1f-4b22-ac40-d4ee45ff8ae1"
  }
}
```

And for the *getOverlays* route:

```
{
  "action": "getOverlays"
}
```

As can be seen from the two examples, not all routes demand the additional content object, this is due to some operations being somewhat simple and therefore not needing the additional information, just as the *getOverlays* route, that returns all overlays found in the *OverlayTable*, and thereby is not dynamic in its response.
This is unlike the *sendAnswerToOverlayContent* route, that needs information on which overlay did the user just respond? Which of the overlay contents is it that the user responded to? What did the user answer? And what user answered this? Where the value for the "answer" key, might also be

an array, if the overlay content allows for multiple selections. And the latter key "overlayCookieId" is used to identify the user, followed by registering that the user has responded to this overlay content, and therefore should not receive it again.

Persistence

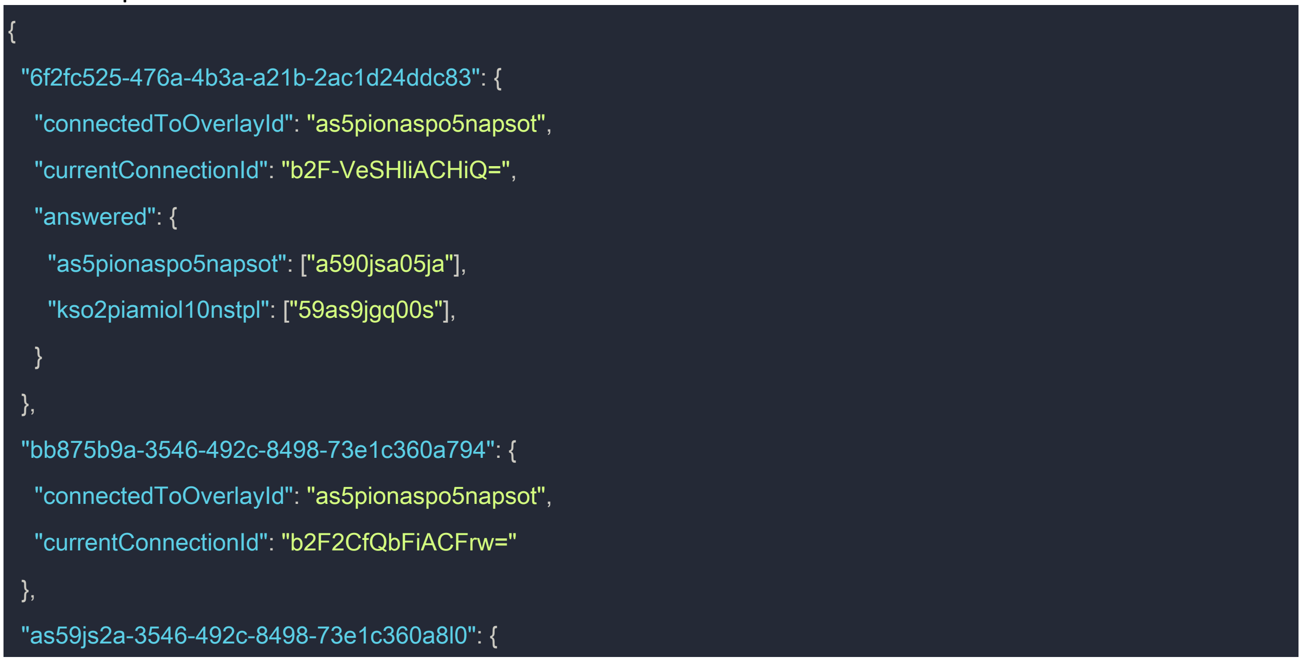
The persistence part is handled by the Amazon DynamoDB service, which contains two tables each with their own responsibilities, but often used in conjunction by the backend; as DynamoDB is a NoSQL database, none of the tables contain any pre-defined keys apart from the obligatory "id".

First, the *ConnectionsTable* contains information about users of the Overlay, however, the information that gets saved about users is minimal and identifying users just from the data saved about a user, would be a near impossible task.
The data saved about each user is shown in Table 5 below.

Key	**Type**	**Description**
connectedToOverlayId	Text String	Shows which overlay the user is watching, this is used to trigger overlay content on all users watching a specific overlay. Will be null if the user currently is either, not watching an overlay or not connected.
currentConnectionId	Text String.	Shows which connection id the user has been allotted, this is used when a message needs to be sent to a user, without the user asking for it first. Will be null if the user currently is not connected.
answered	JSON Object	Contains JSON objects with the same keys as existing overlay ids, these keys contain a list, with each entry being a String of an id of an overlay content, that can be found on that overlay.

Table 5: Keys in ConnectionsTable.

An example on this with three users saved in *ConnectionsTable* can be seen below.



```
{
  "6f2fc525-476a-4b3a-a21b-2ac1d24ddc83": {
    "connectedToOverlayId": "as5pionaspo5napsot",
    "currentConnectionId": "b2F-VeSHliACHiQ=",
    "answered": {
      "as5pionaspo5napsot": ["a590jsa05ja"],
      "kso2piamiol10nstpl": ["59as9jgq00s"],
    }
  },
  "bb875b9a-3546-492c-8498-73e1c360a794": {
    "connectedToOverlayId": "as5pionaspo5napsot",
    "currentConnectionId": "b2F2CfQbFiACFrw="
  },
  "as59js2a-3546-492c-8498-73e1c360a8l0": {
```

```
  "connectedToOverlayId": null,
  "currentConnectionId": null,
  "answered": {
   "as5pionaspo5napsot": ["a590jsa05ja", "ba52sy0as501"]
  }
 }
}
```

To elaborate further on the "answered" key, sometimes it will not exist on a user entry in the *ConnectionsTable*, if that were the case, it would mean that the user in question, has not yet responded to a question, and therefore it will not have get created yet.
This is due to the "answered" key, as the name may allude to, contains which of the overlays that the user has sent an answer to, and therefore should not get prompted to answer again, both because it may be annoying, and as it may skew answer distributions; and if a user has not yet answered just once yet, it is technically a waste of space in the database, to have created the value.
Then lastly, why "answered" is not called "responded", is solely because if the overlay got updated with overlay content, capable of sending questions through as many times as the user wishes to, or whenever they want to, it would not make sense to save this in the answered value, therefore such an overlay content, would not contain an "answer", but possibly an "response", "question" or "message".

The *OverlayTable*, contains all information about an overlay, and where the complete configuration for the overlay content(s) are to be found.
The data saved about each overlay is shown in Table 6 below.

Key	**Type**	**Description**
broadcastTitle	Text String	Contains the title of the broadcast of which the overlay is meant for. Example: "Political party debate 27th of October 2022"
overlayContent	JSON Object	Contains a map of JSON Objects with overlay content that the overlay contains. Each overlay content will then contain: - "answers" ○ A JSON object with contain the answer statistic for the options. - "answerType" ○ A text String with what kind of answer this overlay content accepts, as example "Integer" accepts a single answer, which increments the answer with 1, and "Multiple Integer" accepts multiple, with each answer getting incremented with 1. - "componentName" ○ A text String, with which component the overlay should load to properly show this overlay content to the user.

		- "props" ○ A JSON object containing information regard the options in the overlay content and the title of this overlay content. Will get passed to the relevant component in the Overlay.

Table 6: Keys in OverlayTable.

An example of an overlay saved in *OverlayTable* can be seen in Appendix 1.

Regarding the "answers" key contained in the "overlayContent" key, as with the "answered" key in the *ConnectionsTable*, this would not exist on overlay content that does not take answers per se, but instead be replaced with "responses", "questions" or "messages", depending on the type of response the user can send.

System behavior.

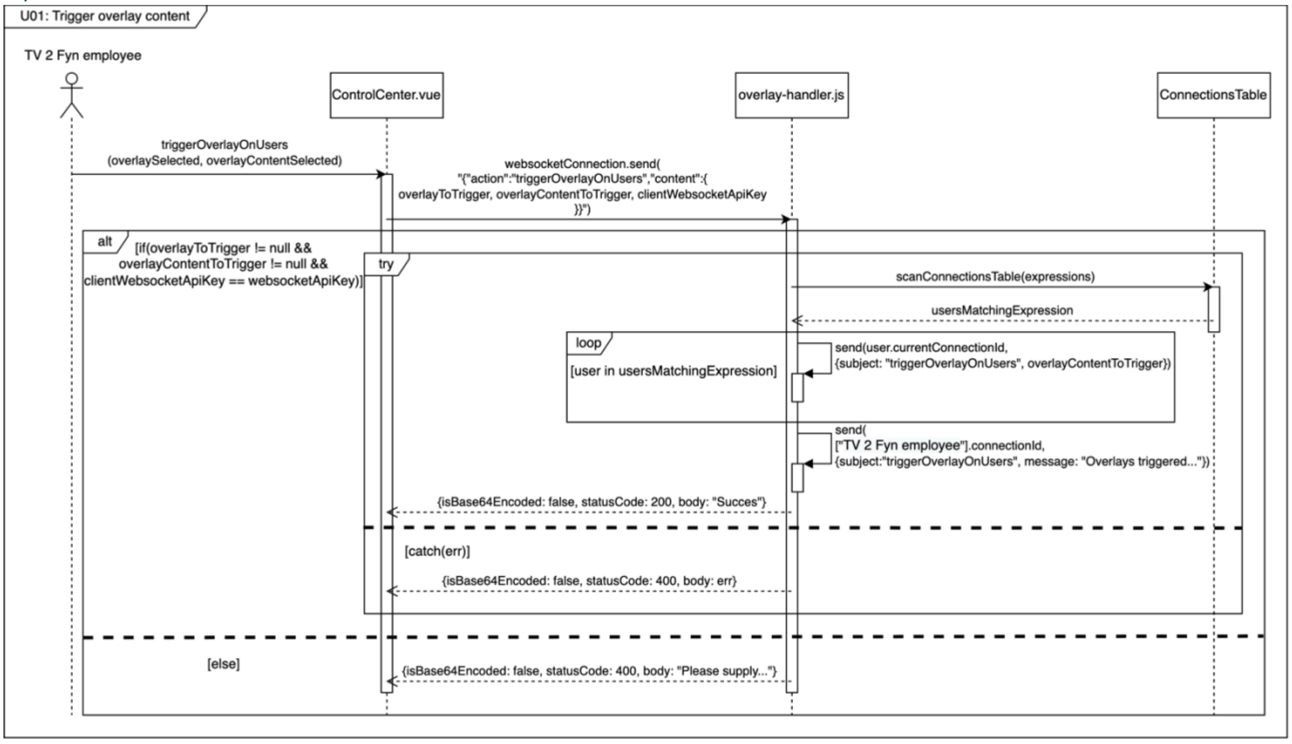

U01: Trigger overlay content
TV 2 Fyn employee
ControlCenter.vue
overlay-handler.js
ConnectionsTable
triggerOverlayOnUsers (overlaySelected, overlayContentSelected)
websocketConnection.send("{"action":"triggerOverlayOnUsers","content":{ overlayToTrigger, overlayContentToTrigger, clientWebsocketApiKey }}")
alt
[if(overlayToTrigger != null && overlayContentToTrigger != null && clientWebsocketApiKey == websocketApiKey)]
try
scanConnectionsTable(expressions)
usersMatchingExpression
loop
[user in usersMatchingExpression]
send(user.currentConnectionId, {subject: "triggerOverlayOnUsers", overlayContentToTrigger})
send(["TV 2 Fyn employee"].connectionId, {subject:"triggerOverlayOnUsers", message: "Overlays triggered..."})
{isBase64Encoded: false, statusCode: 200, body: "Succes"}
[catch(err)]
{isBase64Encoded: false, statusCode: 400, body: err}
[else]
{isBase64Encoded: false, statusCode: 400, body: "Please supply..."}

Figure 3: Sequence diagram for U01: Trigger overlay content.

Figure 3 shows the flow of the program when a *TV 2 Fyn employee* wishes make use of "U01: Trigger overlay content", it triggers when the employee clicks on the button associated with the *triggerOverlayOnUsers()* function. The Control center (*ControlCenter.vue*), then sends a message to the backend route "triggerOverlayOnUsers" with a body containing which overlay to trigger(overlayToTrigger), what overlay content to trigger(overlayContentToTrigger), along with an API key(clientWebsocketApiKey); if the 2 first values are not null, and the API key correct, the flow sequence continues otherwise an error message is sent back; it is important to note that if the value in "overlayToTrigger" does not point to an existing overlay id, no users will get found later on, and if the value in "overlayContentToTrigger" does not exist on the overlay referred in "overlaySelected", nothing will happen when a user receives the trigger message.

On the background of the values passed, the overlay-handler.js Lambda file, will get all the users that are registered as currently watching the “overlaySelected”, alongside filtering out all users that has already answered “overlayContentSelected”, the users with the right values will then get a sent a trigger message each, containing which overlay content that should trigger/show the overlay content (overlayContentToTrigger).
Following this the Control center, which is running in the web browser of the *TV 2 Fyn employee,* gets back a message stating that the operation was a success, followed by a WebSocket response telling the web browser, that the operation was a success.
If any error happens during the operations performed on the backend (*overlay-handler.js*), an error response is sent back to the WebSocket client in the Control center.

Iteration planning

1st iteration

2nd iteration

….. iteration

Implementation

1st iteration

2nd iteration

….. iteration

Test

Testing the Non-functional requirements

Too test whether the system lives up to the Non-functional requirements, “NF01: Usability” and “NF02: Performance”, a series of test were conducted on the system.

Testing Usability of the Overlay

To test the usability of the system, and in this case specifically the overlay, a series of test questions were devised, which should give a pretty good view of whether the overlay lives up to the Usability requirement, and what users of the Overlay might wish from a future version of it.
In total, 10 people performed the test, which consisted of 4 family members and 6 employees at TV 2 Fyn, they were presented with the questions following using the Overlay, without any prior introduction to it, and without getting told what would happen during usage of the Overlay.
. The test questions devised can be seen below in Table 7.

ID	**Question**
UQ01	Do you like the overlay? Is it easy and intuitive to use?
UQ02	What would you have done differently with the design of it?
UQ03	Does it interfere with the video?
UQ04	Any feature(s) you are missing?

Table 7: Test questions for the usability test.

Alongside with the four test questions, the test subjects were also asked where they would prefer the Overlay to be placed on the screen, the pictures can be seen in Appendix 2.

Placement of overlay.

Answer	**Count**	**Percentage**
Top left	2	20%
Top right	0	0%
Top middle	0	0%
Top wide	0	0%
Bottom left (Current overlay placement)	6	60%
Bottom right	2	20%
Bottom middle	0	0%
Bottom wide	0	0%

Table 8: Test answers for overlay placement test.

The first test that the test subjects went through, asked where on the screen they preferred the overlay to placed, and to make sure that they current design of the Overlay did not affect their answers, this test was done before they saw the Overlay “live”. The purpose of the test was to see whether the current placement configuration should change in the future.

From the test, the answers shows that luckily the Overlay suits the majority, with each of the two different answers that got selected, sharing at least one attribute with the current configuration (in the bottom, or to the left). Therefore, it could be said that none of the other placements, would have made sense. However, a feature giving the User a possibility to customize the placement might make sense to implement in the future.

UQ01 - Do you like the overlay? Is it easy and intuitive to use?

Answer	**Count**	**Percentage**
Yes	8	80%
No	2	20%

Table 9: Test answers for UQ01

The purpose of this test was to see whether the Overlay lives up to the goal of being usable without an introduction.

The result of this test, shows that the Overlay was easy and intuitive to use for 80% of its' users, the last 20% did not think so though, which is to be expected, as they are of an older generation (80+), and might not be as used to navigating a computer, and might also not be the target group of the Overlay, as they would probably stick to watching TV broadcasts through the TV, and not use a computer, or phone for it; however, it may make sense to figure out how to guide those who might find the Overlay hard to use.

UQ02 - What would you have done differently with the design of it?

Answer	**Count**	**Percentage**
Nothing/ I do not know	5	50%
Make it clearer when you can pick multiple options	4	40%
Different colors	2	20%
Larger text	2	20%
It is a little bit "boring"/plain	1	10%

Table 10: Test answers for UQ02.

The purpose of this test was to see whether the design of the Overlay lacks something major in its' user interface, or if something should get added or removed to increase the user experience. Everyone could give multiple answers/ feedback points, to this question.

In this test, some good ideas actually came forth from the answers, such as making it clearer when it is possible to pick multiple options, which 40% mentioned, as it may not be super clear for all, because it right now is designed in a way, where it is expected that the user knows that a Circle = Single selection, and Square = Multiple selections (see Figure 4 below).

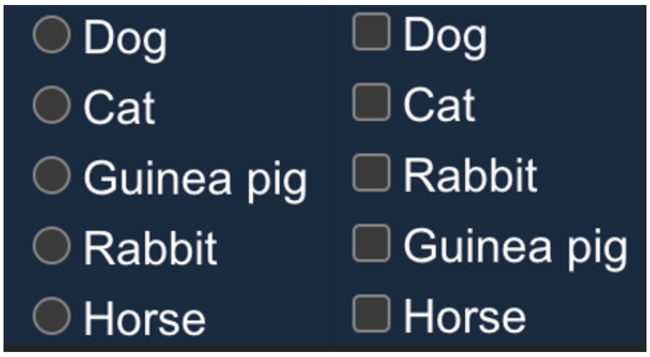


Figure 4: Example of design for single and multiple selection. Left = Single selection, Right = Multiple selections.

On top of this, 20% wished for different colors in the design, which could definitively be an option, but as of right now the colors have been chosen based on the color schema used by TV 2 Fyn; it may though be a possibility to add functionality for letting specific overlay content control the colors, and probably make it a custom prop that can be defined when creating overlay/overlay content.

Like with the colors, 20% wished for larger text, which coincided with being the same 20% whom answered "No" in UQ01, which may again be a product of their older age, which sometimes is correlated to a worse vision, that may lead to the wish for larger (more readable) text.

With regards to the 10% thinking that it is a bit "boring"/plain, it is somewhat the plan, as the focus has been on simplicity, so that it does not steal all the attention from the video, while being a insignificant load for the web browser.

UQ03 - Does it interfere with the video?

Answer	Count	Percentage
No, not at all	9	90%
A little bit	1	10%

Table 11: Test answers for UQ03.

UQ04 - Any feature(s) you are missing?

Answer	Count	Percentage
Button to close it like with a web browser	6	60%
Possibly to send text questions/messages	4	40%
Button to minimize it	3	30%

Table 12: Test answers for UQ04.

Test Performance of the system

To test the performance of the system, a test were set up by setting up by printing the current time when the *triggerOverlayOnUsers()* function were run in the Controler center, and then again when the Overlay received a message with the subject: "triggerOverlayOnUsers"

Control center sent	Overlay received	Difference
1670597698780	1670597698974	194 ms
1670597833657	1670597833857	200 ms
1670597842515	1670597842694	179 ms
1670597848726	1670597848915	189 ms
1670598728656	1670598728917	261 ms
1670598775771	1670598776088	317 ms
1670598887010	1670598887293	283 ms
1670598887010	1670598887257	247 ms
1670598961110	1670598961355	245 ms

1670598961110	1670598961334	224 ms

Table 13: Performance test scores.

Lowest: 179 ms.
Average: 234 ms.
Highest: 317 ms.

Discussion

Conclusion

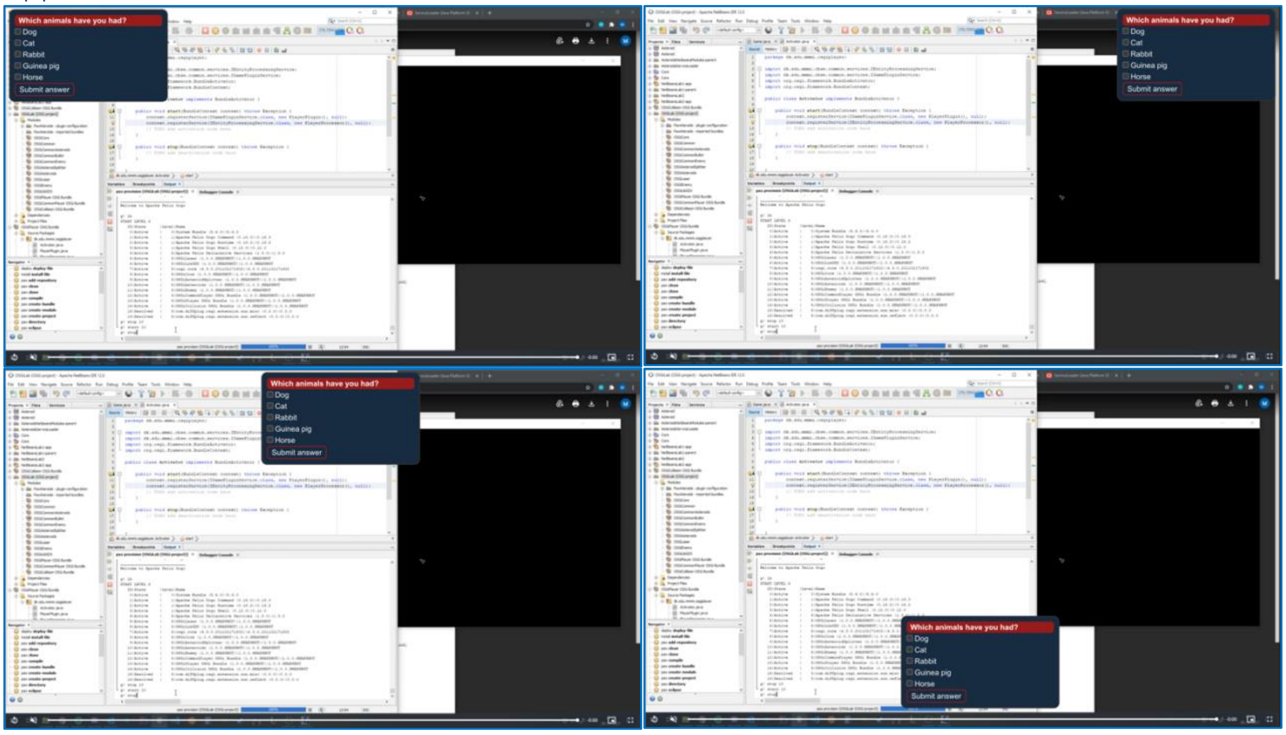
Process evaluation

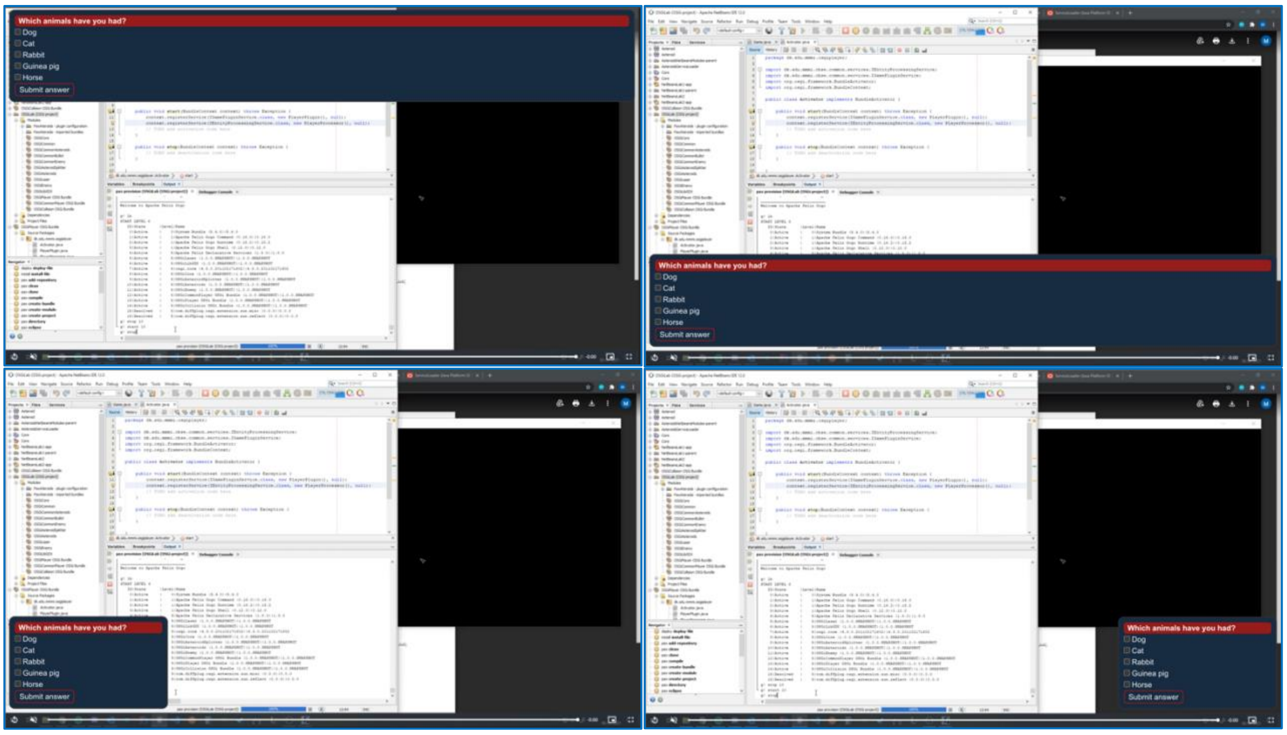
Appendix

Appendix 1

```json
"908awh98ahs689as896": {
  "broadcastTitle": "Some broadcast regarding the election",
  "overlayContent": {
    "a5s0a0gmjs0aw9": {
      "answers": {
        "sv": {
          "amount": 0
        },
        "svi": {
          "amount": 0
        },
        "svø": {
          "amount": 0
        },
        "sø": {
          "amount": 0
        },
        "vi": {
          "amount": 0
        }
      },
      "answerType": "Multiple Integer",
      "componentName": "OverlayMultiSelect",
      "props": {
        "options": [
          {
            "label": "SV",
            "value": "sv"
          },
          {
            "label": "SVI",
            "value": "svi"
          },
          {
            "label": "VI",
            "value": "vi"
        },
        {
          "label": "SØ",
          "value": "sø"
        },
        {
          "label": "SVØ",
          "value": "svø"
        }
      ],
      "title": "Which constellations would you like?"
    }
  },
  "a8sh58ah5": {
    "answers": {
      "enhedslisten": {
        "amount": 0
      },
      "liberal_alliance": {
        "amount": 0
      },
      "socialdemokratiet": {
        "amount": 0
      },
      "venstre": {
        "amount": 0
      }
    },
    "answerType": "Integer",
    "componentName": "OverlaySelect",
    "props": {
      "options": [
        {
          "label": "Enhedslisten",
          "value": "enhedslisten"
        },
        {
          "label": "Liberal Alliance",
          "value": "liberal_alliance"
        },
        {
          "label": "Socialdemokratiet",
          "value": "socialdemokratiet"
        },
        {
          "label": "Venstre",
          "value": "venstre"
        }
      ],
      "title": "Which political party won the debate?"
    }
  }
 }
}
```

Appendix 2




Which animals have you had?
Dog
Cat
Rabbit
Guinea pig
Horse
Submit answer