Western Norway University of Applied Sciences

BACHELOR'S THESIS

Live Video

Steffen Lid Gaustad

Nicolas M. Mjøs

Information Technology & Computing
Department of Computer science, Electrical engineering, and Mathematical sciences
Supervisor Sven-Olai Høyland

04.06.2021

Western Norway University of Applied Sciences

Faculty of engineering and science

Department of Computer Science, Electrical Engineering and Mathematical Sciences

TITTELSIDE FOR HOVEDPROSJEKT

Rapportens tittel: Live Video	*Dato:* *04.06.2021*
Forfatter(e): Steffen Lid Gaustad og Nicolas Marchant Mjøs	*Antall sider u/vedlegg:* *23* *Antall sider vedlegg:* *3*
Studieretning: Informasjonsteknologi og Dataingeniør	*Github Repository:* https://github.com/vizstory/livevideo
Kontaktperson ved studieretning: Sven-Olai Høyland	*Gradering:* Ingen
Merknader:	

Oppdragsgiver: Vizrt	*Oppdragsgivers referanse:* Mikal H Henriksen
Oppdragsgivers kontaktperson: Simen Nytun	*Telefon:* +47 48144275

Sammendrag:

Bachelorprosjektet går ut på å lage en prototype som gjør det enkelt å sende direkte video inn til ett TV studio fra smartenhet over mobilt bredbånd.

Summary:

The Bachelor project aims to make a prototype that makes it easy to send live video to a TV studio from a smart device over mobile broadband.

Stikkord:

SRT	Streaming	Media

Høgskulen på Vestlandet, Fakultet for ingeniør- og naturvitskap
Postadresse: Postboks 7030, 5020 BERGEN Besøksadresse: Inndalsveien 28, Bergen
Tlf. 55 58 75 00 Fax 55 58 77 90 E-post: post@hvl.no Hjemmeside: http://www.hvl.no

Western Norway University of Applied Sciences

Faculty of engineering and science

Department of Computer Science, Electrical Engineering and Mathematical Sciences

PREFACE

We would like to thank Vizrt for providing an interesting challenge as well as guidance, software and particularly Simen Nytun our development point of contact.

The rest of the team has also been integral to the development of the project:

Mikal H Henriksen	- Project Manager
Øyvind Neuman	- Backend development resource until end of March 2021.
Gisle Sælensminde	- Transcoding and protocols.
Knut Arvidsson	- Coordination and infrastructure.
Roger Rebbestad Sætereng	- Legal issues, contracts, and accounting.

We are also grateful for the guidance provided by Sven-Olai Høyland, our HVL supervisor.

Steffen and Nicolas

TABLE OF CONTENT

1 INTRODUCTION

1.1 Motivation and goal

The motivation for this project is to make it easier, quicker and cheaper to broadcast remotely. The desired result is to make content producers more independent and flexible, as well as reducing the time it takes to release a story. If people had a quick and simple way to do video streaming from smart devices, they would be able to publish stories quicker which is a huge advantage in today's fast-paced social media world.

The goal is to make a product with which customers will be able to start broadcasting live video from anywhere, with nothing but a smart device like a phone and a 4G internet connection. The stream will be routed into a production studio where editors can prepare the content and publish it as quickly as possible.

1.2 Context

In today's media landscape being quick to report on current events is important. Providing a solution for media institutions to quickly ingest new content, manage it, produce it and then broadcast to several platforms in an efficient manner can be a game-changer. Currently, the project owner provides a video software suite that can ingest local content, produce, manage and broadcast to fit several different workflows. The software suite lacks a user-friendly way for journalist to provide instant live content from remote locations to be used in these workflows.

TV studios currently spend a lot of time and resources managing remote broadcasts. In order to broadcast video from a remote location they require a lot of equipment and personnel to set it up and operate it. Due to the explosion of social media, often viewed on small screens like phones and tablets, requirements for production quality are lower than they used to be. Furthermore, smart devices have better cameras and more powerful hardware than they used to have. For these reasons, the focus has shifted towards release time and content rather than quality. This makes streaming an attractive option as it allows extremely quick release time with acceptable compromises on quality.

1.3 Limitations

The primary limitations of this project are time and knowledge. The technology available makes a solution possible. However, arriving at a design is not simple. The most important limiting factors are as follows.

The SRT (Secure Reliable Transport) protocol uses UDP (Haivision, 2021) which means it is not possible to transmit SRT streams from browsers. This is caused by the fact that browsers do not allow sending UDP packets (Fiedler, 2017). A consequence of this is that relying on SRT also means that a native application is required on the client-side.

Another limiting factor is that the video stream will end up on a private network. This has implications for how the system is built, as it means the data stream cannot be transmitted directly to its destination. Rather, it needs to go through some sort of gateway into the

private network. For this reason, a middleman is required between the public Internet and a customer’s private network.

Finally, time and knowledge are large factors. There are strict deadlines, meaning it is important to plan well and be conservative when deciding which features to prioritize. This matter is made worse by the research requirements, and much time has been spent learning about tools and testing various solutions.

1.4 Resources

The most important and unique resource for this project is the Project Owner’s research team. They have invaluable experience with the technology this project revolves around and are also supplying their own products for use in the project. Vizrt’s transcoder software is going to be an important part of the project and the team can provide guidance through a Microsoft Teams channel as well as video meetings.

Beyond this, the Internet is an extremely valuable resource. All tools used in this project have extensive documentation available online, and this is used frequently during the research and development phases of the project.

2 PROJECT DESCRIPTION

2.1 Practical background

2.1.1 Project owner

The project owner is Vizrt. Vizrt is a large Norwegian company founded in 1997. It has its headquarters in Bergen and is currently operating 30 offices worldwide with over 700 employees. (Vizrt, 2021) A quote has been selected from their website, to describe what they do and who their customers are:

> *"Vizrt is the world's leading provider of visual storytelling tools for media content creators in the broadcast, sports, digital and esports industries*
>
> *Vizrt offers market-defining software-based solutions for real-time 3D graphics, video playout, studio automation, sports analysis, media asset management, and journalist story tools.*
>
> *Vizrt's promise is to master complexity and maximise creativity.*
>
> *More than three billion people watch stories told by Vizrt customers everyday including from media companies such as CNN, CBS, NBC, Fox, BBC, BSkyB, Sky Sports, Al Jazeera, NDR, ZDF, Network 18, Tencent, and many more.*
>
> *Vizrt is part of the Vizrt Group along with its sister brands, NewTek and NDI. Vizrt follows the single purpose of this Group; more stories, better told."*
>
> *(Vizrt, 2021)*

The project owner has provided the project idea, the transcoder used in the project, the time of their colleagues and would under normal circumstances have provided office space as well. However, due to the Covid-19 pandemic it has been necessary to work from home and use video meetings for communication.

2.1.2 Previous work

Secure Reliable Transport

SRT (Secure Reliable Transport) is a relatively new protocol, developed by Haivision with the goal of providing Secure and Reliable Transport of data. It is also highly efficient.

SRT Reference Implementation

The Haivision SRT reference implementation (Haivision, 2021) is used for secure and reliable media streaming over the internet. While it is possible to use the library to develop entirely custom solutions, it was found that an example tool named srt-live-transmit.cpp included all the necessary functionality.

Web-RTC

Web-RTC (Real-time communication for the web) is a widely supported, open standard for data transmission in browsers. (Google, 2021)

Coder

Coder is a proprietary transcoding software developed and maintained by Vizrt. It exposes an API to allow control of its transcoding jobs and contains built-in documentation. This documentation does not appear to be publicly available and as such is not included as a source in this report, even though it has been used extensively during the development phase of the project.

Network Device Interface

NDI (Network Device Interface) is a standard developed by NewTek that allows ultra-low latency video on existing IP video networks (NewTek, 2021). This is already in use among the Project Owner's clients and the reason for using Coder – to transcode video signals from SRT to NDI.

2.1.3 Initial requirements specification

The primary requirements set forth by the project owner are as follows:

- Secure and reliable high-definition streaming
- Mobile device prioritization
- Acceptable latency and quality on lower bandwidth unstable connection(3G)

The essence of this project is to develop a solution that enables live streaming of video from a mobile source such as a smartphone to a server on a private network running NDI. Due to the requirement of connecting to a private network, it is necessary to develop a "middle-man" server that can serve as a bridge between the internet and the private network. It is also necessary to transcode the video stream from some other streaming protocol to NDI because the NDI protocol is not suited for streaming over unreliable networks. Transcoding will be handled by Coder running on the private network.

2.1.4 Initial solution idea

The chosen solution is to develop a server that can receive and forward SRT data streams. The server will forward the received streams to a transcoding server on a private network. A stream can be initiated from any software capable of producing SRT streams, it will then go via this server to Coder where it is converted to NDI and published on a private NDI system.

The primary goal of this solution is to serve as a prototype, demonstrating how easy it can be to stream video from a smartphone. The design of the application will focus on automating as many steps of the process as possible, with little regard to things like accessibility or look and feel.

3 PROJECT DESIGN

3.1 Possible approaches

Multiple approaches have been considered. One major consideration is which streaming protocol to use. The two main contenders are Web-RTC and SRT. Web-RTC is somewhat old, well established and browser-based which is a big benefit for this use case. However, it is also less reliable and less capable of dealing with bandwidth issues. This is where SRT shines. It promises better quality, security, and reliability with the caveat that SRT uses the UDP protocol which is not supported by modern browsers due to security concerns. This means that in order to use SRT, the user needs to install a native application on the streaming device, as opposed to streaming through a browser.

One possible solution is to use Web-RTC and have the entire project be browser-based. The user would log into a website and initiate their stream directly from that website. This solution seems fairly elegant and simple but would be bound by Web-RTC's limitations.

Another possible solution is to use SRT along with a native application. This application could be an existing third-party application, or it could be custom developed as part of the solution. Due to the time constraints, it seems more realistic to use a third-party application. For this, Larix Broadcaster (Softvelum, 2021) seems to fit the use case very well. This means the user needs to have either an Android or iOS device for the prototype. Since mobile devices are a priority and Android together with iOS have above 95% of the market share (S. O'Dea, 2021) this was not an issue for the employer.

Finally, the client device needs to have information on where to send the stream. If the streaming client is running in the browser, this would be simple as the information should be available. If the client is a native application, the info needs to be passed to it somehow. Larix broadcaster allows the opening of URI shortcuts and even provides a tool to make them; Larix Grove (Softvelum, 2021). To provide working URIs, the web application will have to generate them on the fly.

3.1.1 Nimble Streamer as a media tunnelling server

Nimble Streamer (Softvelum, 2021) advertises as a freeware media server supporting SRT among a large number of streaming protocols. Configurable by WMSPanel, a control web panel sold by Softvelum, the capability of the software seems to align with the needs of the solution. Using the WMSPanel API (Softvelum, 2021) most of the video streaming capabilities of the solution could be handled. However, since it is a paid service, this is not a good solution for the project owner.

Softvelum also provides what they call a native API for Nimble Streamer (Softvelum, 2021) which seems like it should be sufficient. This API unfortunately does not have full Nimble Streamer functionality the way the WMSPanel API has. It is possible to reload the config files (Softvelum, 2021) through the API. A possible solution would then be to write to the config files with a local application and reload them with the API call. This would require reverse engineering of the config files which are not documented and as such has not been considered a viable option.

3.1.2 Building our own media tunnelling server with SRT

Multiple approaches to using the SRT library indirectly have been explored. There is a Node.js library named @eyevinn/srt (Eyevinn, 2021) by a Swedish company named Eyevinn which looked promising but proved challenging to install.

Another library that was considered is SrtSharp (Cinegy GmbH, 2021) by Cinegy GmbH. This library was eliminated due to being incomplete and very poorly documented.

The last solution is to use the SRT library directly. For this solution, there are two possible approaches: Create a C++ program that uses the library or create a binding to allow usage of the C++ library from a different language such as C#. There are also example programs provided in the library which may be suitable to use directly or modify. A working proof of concept using one of these example programs has been successfully tested, making this seem like a promising approach.

3.1.3 Building our own media tunnelling server with NGINX

"NGINX is open-source software for web serving, reverse proxying, caching, load balancing, media streaming, and more" (NGINX, 2021)

Using NGINX gives access to RTMP (NGINX, 2021) a module and a lot of documentation to build a capable media server. It even features an API module (NGINX, 2021) which would make building the web application faster. Unfortunately, NGINX does not yet support SRT which is the preferred protocol. Developing a custom SRT implementation in NGINX is possible. However, until the SRT portion is solved becoming familiarized with NGINX is a big task.

3.1.4 Discussion of alternative approaches.

Considering that this product is intended for the media, video quality and reliability are important factors. Security is also a high priority, and for these reasons, the SRT streaming protocol is preferred. Being able to support other protocols could also be beneficial, but this would take more time and complicate the project as well as the user experience.

A lot of time was spent exploring the option of using Nimble Streamer. In the end, it was decided that it does not provide sufficient alternatives for control without the added cost of WMSPanel and it is not desirable to be reliant on a paid third-party program so Nimble is no longer an option.

The option of using Node.js with the @eyevinn/srt library could prove beneficial. It would allow usage of the vast number of Node.js modules available, allowing quick and convenient development. Research on this was stopped due to technical issues but it may be explored further in the future.

A proof of concept was constructed using Larix Broadcaster on a smartphone, the srt-live-transmit.cpp app included in the SRT library, Vizrt's transcoder and NDI. Video streaming was successfully performed from a smartphone to the NDI system. This leaves the option of using the SRT library directly with an advantage, as it is proven to work. Another option has also been considered – using the srt-live-transmit.cpp application directly through a command line is possible and may be the most ideal solution for the prototype. This seems like a fairly good solution as it keeps the responsibilities separate, allowing the

project owner or someone else to easily implement their own SRT module or otherwise modify the application.

3.2 Specification

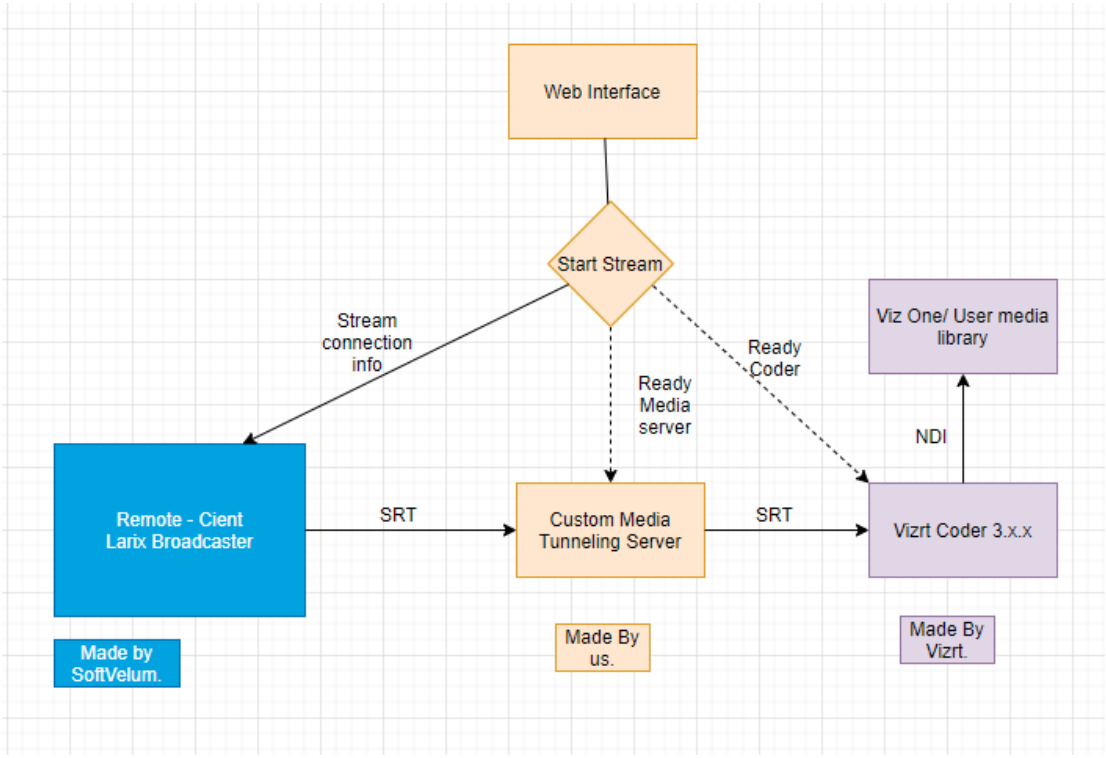

Figure 1: Initial design

The above diagram illustrates the initial plan for the project. There will be a web application managing the system. When a user wants to start a stream, they will visit the web application and perform some guided actions to initiate streaming. The web application will provide the necessary data such as URI and possibly authentication for the client application (Larix). It will contact the backend server to set up a tunnel, and simultaneously contact Vizrt Coder to start a transcoding job. The user can now start streaming and the stream will be made available through the NDI system.

3.3 Selection of tools and programming languages

This section contains an overview of some of the more important tools used for this project as well as a brief explanation of what they are used for.

3.3.1 Encouraged by project owner

These tools were chosen in part because Vizrt already uses them in their workflow.

- Git/GitHub for version control and sharing.
- Microsoft Teams for communication.
- NDI for testing.
- Viz One Coder for transcoding.

3.3.2 Chosen due to familiarity on student team

These tools were chosen due to a mix of previous experience, advice, and a desire to work with them.

- Postman for network diagnostics and testing.
- TypeScript as the primary programming language.
- Node.js with express.js for the backend.
- React for the frontend, with Material-UI to provide premade components.

3.3.3 Project specific tools

These tools are an integral part of the solution.

- LibSRT for the srt-live-transmit application.

A universal data transport tool intended to transport data from SRT to other mediums as well as SRT to another SRT. This is described as a sample tool, but it is enough for our solution.

- Larix Broadcaster for client streaming.

Client for iOS or Android with support for many streaming protocols and recording simultaneously.

3.4 Project development method

3.4.1 Development method

The chosen development method for this project is AGILE. The AGILE method consists of working relatively short sprints with a perpetually working product and working closely with the customer or in this case project owner. Small incremental goals are set frequently, and the focus is on adding small bits of functionality without large sweeping changes.

In the beginning, focus is on the key bits of functionality that the solution needs, with the hope of adding stretch goal functionality later in the development cycle. Since only two students working on the solution, it is important to manage expectations and plan conservatively. Planning too many features may make it difficult to prioritize.

It also seems unnecessary to make a very detailed plan. There are a lot of unknown factors, so it is important to have backup ideas, but for a prototype it is difficult to plan well as the team has little experience with most of the tools used. For this reason, the approach is a relaxed AGILE-like workflow where a working copy of the product is maintained in the main branch while features are added in new branches and only merged once they have been thoroughly tested. This ensures that the project is kept in working order while it is under development.

3.4.2 Project Plan

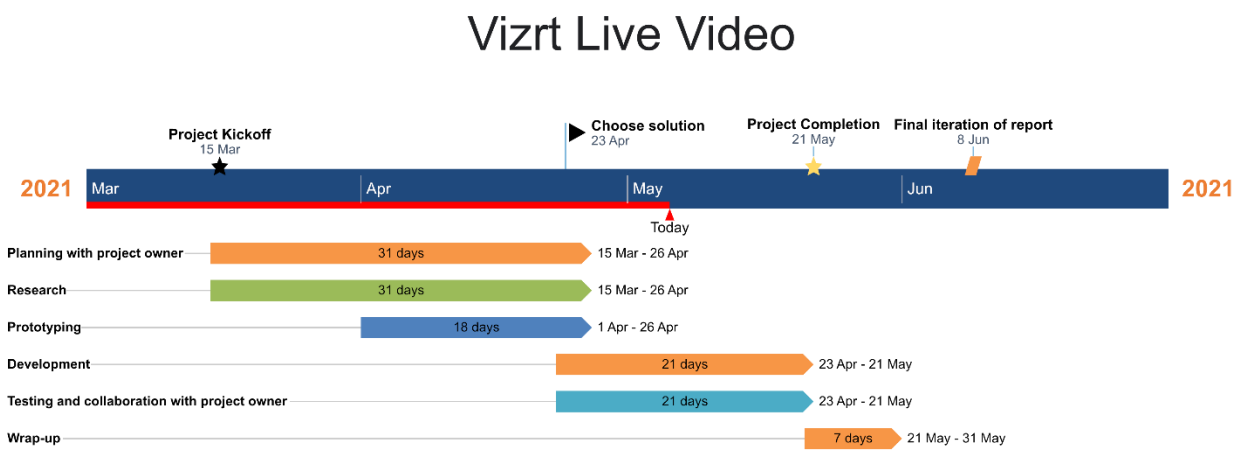

Figure 2: Gantt Diagram

Planning with project owner

Fleshing out the project itself and discussing various approaches etc.

Research

Finding potential new approaches and learning more about chosen approaches.

Prototyping

Testing various approaches and evaluating which ones to pursue further.

Development

Picking a solution and making it work.

Testing and collaboration with project owner

Testing the product and making sure the project owner is happy with it.

Wrap-up

Taking some time to finish everything up so that the project is well documented and potentially useful to the project owner or someone else in the future.

3.4.3 Risk management

Some of the initial values were adjusted as we gained more knowledge about streaming and Vizrt's Coder.

Risk	Likelihood (1-5)	Severity (1-5)	Risk Level (1-25)	Management
Not enough knowledge about streaming protocols.	4→**2**	**5**	20→**10**	Use resource channels, public forums, Vizrt advisors, School advisors.
Streaming solution is not stable enough.	**4**	**3**	**12**	Further testing of different resolutions, framerates, audio bitrates etc.
The solution does not integrate well with Vizrt's workflow.	4→**2**	**3**	12→**6**	Keep up communication with Vizrt, early prototypes will help visualize solution between team members.
Solution does not handle firewalls and port traversal well.	3→**2**	**4**	12→**8**	Use resource channels, public forums, Vizrt advisors, School advisors.
Tools used in solution are revealed to not be sufficient.	4→**2**	**5**	20→**10**	Re-evaluate with team alternate tools we can use, discuss with Vizrt.
Team is affected by illness (Covid-19)	**2**	**4**	**8**	Follow health protocols from state/school. Manage time well to prepare.

*(Table 1: Initial Risk 12.03.2021, **Adjusted Risk 18.04.2021**, the lower the number the better)*

3.5 Evaluation method

Project owner has specified that internal testing of functionality is sufficient for the prototype. Any further testing is out of the scope of the project. Internal testing will include testing with the project owner, as well as a test procedure intended to ensure that testing remains consistent and covers all functionality.

3.5.1 LIVE VIDEO TEST PROCEDURE

In order to ensure that all implemented features are working and continue working properly through various changes, a formalized test procedure has been developed. This procedure details the steps necessary for testing, both from the perspective of the dev team and from the perspective of the tester. The intention is to document the steps required for preparing a test, as well as the steps required to sufficiently test all features. This is to

ensure that all features are sufficiently tested after changes have been implemented, and safeguards against human error and similar issues.

PREPARATION DEV TEAM:

Start application
Create a few streams

PREPARATION TESTER:

Install Larix

PROCEDURE:

1: Open website
Assertion: Existing streams should load quickly

2: Create new stream
Assertion: New stream should appear in streams list

3: Click QR-Code
Assertion: Code should enlarge on first click and shrink on second click.

4: Start streaming by clicking the Larix button, scanning the QR code or manually copying the link into Larix and making sure the settings are correct.
Assertion: Stream should appear in Studio Monitor.

5: Stop streaming, edit the stream name, start streaming again
Assertion: Stream should appear again in Studio Monitor under an edited name.

3.5.1.1 NETWORK TEST PROCEDURE:

Check that changes made on one client is reflected across other clients.

4 DESIGN AND CREATION

4.1 Redesigning the frontend and backend

In the initial specification, the plan was to have the frontend communicate directly with Coder. However, as the configuration was set up with a node express backend and a node react frontend it was decided to change this approach. The initial idea was that the SRT Transmit component should be as cleanly built as possible, only being responsible for a single task. However, React runs in the client's browser, having a direct connection to Coder could present security concerns. Therefore, the new specification has the SRT Transmit part only as a component of the backend server with the rest of the network hidden from the frontend.

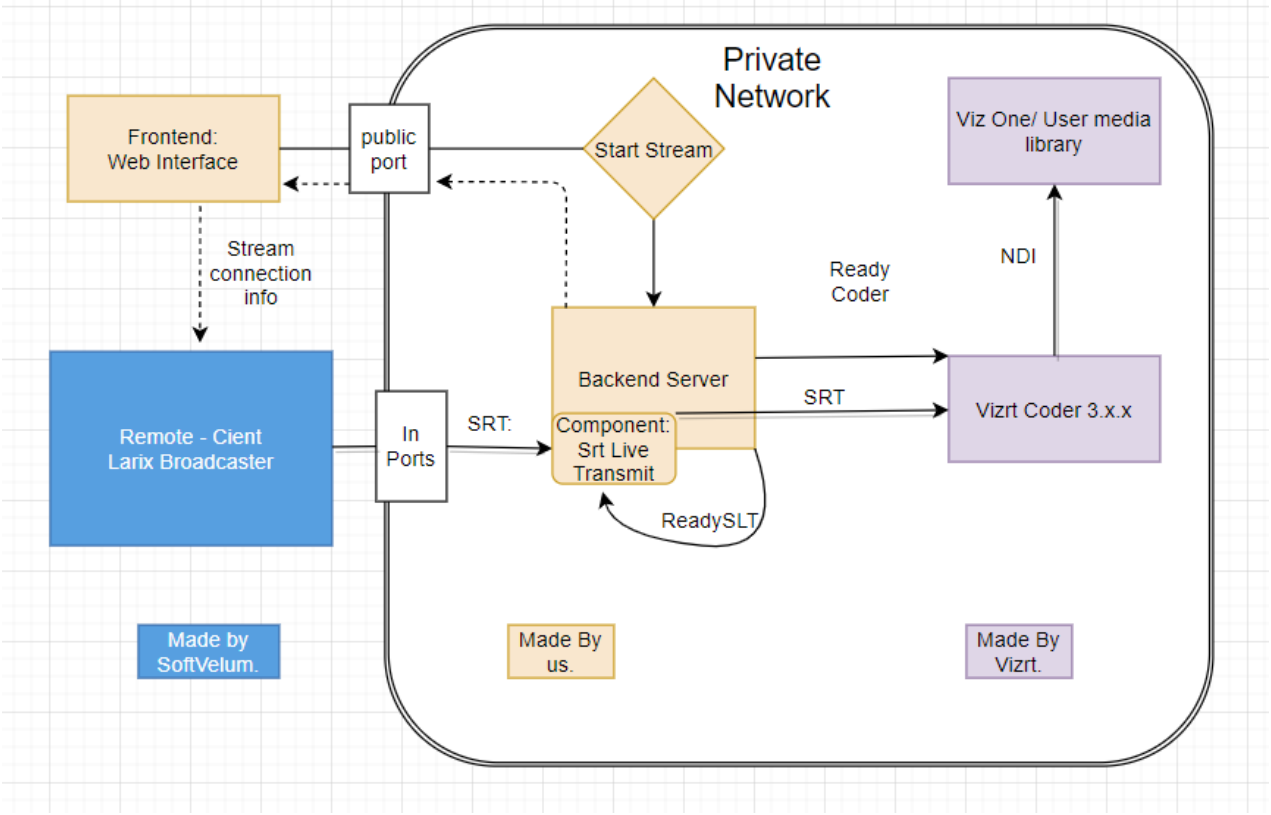

Figure 3: Final Solution design

4.2 Project configuration

The backend and frontend are both Node.js projects configured within the same repository with a common proxy. Node also makes dependency easy as after the install command is run anyone else using the repo simply must run "npm ci" to install them. The configuration contains scripts allowing them to run separately or concurrently depending on which features currently under testing.

4.3 The Backend Server

This section describes the component referred to as “the backend”. The backend is a Node.js application that is responsible for all the actual functionality of the application. Upon receiving messages from the frontend, it can start new instances of srt-live-transmit and new jobs in Coder. It is also responsible for keeping track of active transmissions, updating the frontend if there are changes in a transmission’s status either on the srt-live-transmit side or the Coder side.

4.3.1 StreamManager and API

The Backend server consists of an express app which through URL paths, the backend API, communicates with an instance of our StreamManager class. The StreamManager is responsible for holding the state of the app, keeping a list of StreamHandler instances which handle individual streams. The StreamManager uses the config to give each Stream a public input port and a private coder port and subscribes to eventdispatchers in the Streamhandler classes. Each StreamHandler also holds an instance of SLTHandler and CoderHandler, the former handling the precompiled SLT process and the latter handling communication with the Coder server through the API.

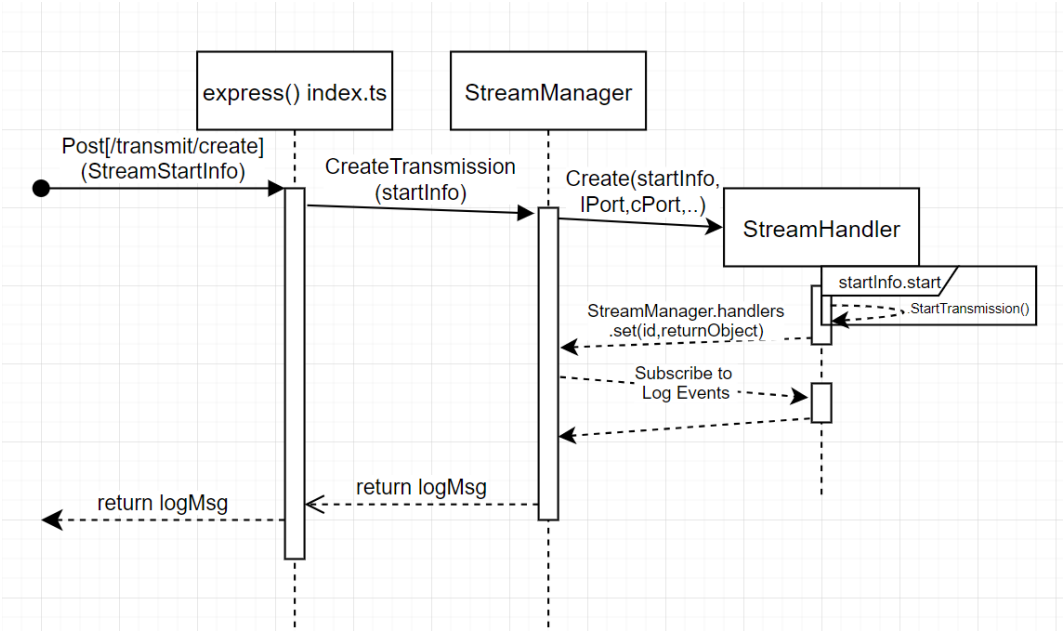

Figure 4: Sequence diagram, starting a transmission.

4.3.2 Handling SLT

The SLTHandler class holds info on how to spawn the process and on StartTransmission() creates the child process. To control the precompiled SRT-Live-Transmit program we run an instance using Node’s “spawn” from “child_process”. To monitor exceptions, crashes, or stream info we parse a combination of readable streams on the spawned child process. For example, to monitor connection status we use a readline interface on “stdout” to read line by line and look for relevant lines like “SRT target connected”. Changing settings is done by killing the process and

running it again with the relevant parameters. Normally to run it in windows CMD you would do something like:

```
path/srt-live-transmit -v
"srt://10.0.0.55:6754?mode=listener&latency=800&maxbw=200000"
srt://10.0.0.55:6756?mode=caller
```

Using node this becomes:

```
let slt = spawn(“path/srt-live-transmit”, ["-v",
“srt://10.0.0.55:6754?mode=listener&latency=800&maxbw=200000"
,”srt://10.0.0.55:6756?mode=caller”]);
```

As seen, spawn takes a path argument to the program being executed, then an array of strings which are the arguments. The search parameters, everything behind the question marks, are added to the URI’s to specify SRT settings. These settings are specified in the SRT Live transmit documentation (Haivision, 2021).

Running an external program using user defined parameters could be considered a security risk. Therefore, we chose to use Node’s “spawn” instead of something like “Exec”. “Spawn” is not executed in a shell and is therefore not as vulnerable to exploits and not susceptible to command injection. At most the chosen program will run with unexpected behaviour.

4.3.3 Handling the coder connection

Coder has a rest API that uses XML, Atom syndication and their own specification VCOS. Deciding to keep things simple we do not use the full Atom syndication feed, only making specific GET request, lightly parameterized POST request and parsing specific entries in the response using an XML utility. The parameters are received from the frontend and must match profiles in Coder to work properly. Coder jobs are created on Stream start and deleted on Stream stop, this may change further down the line but for now, it keeps things simpler and avoids some Coder quirks.

4.4 Frontend

The Frontend is built with React using Materials UI (Material-UI, 2021) as an asset library. The focus of the UI was simplicity, as it is made for non-technical users. Starting a stream requires one to simply enter a preferred stream name, and if needed a specific streaming profile. Creating the stream automatically starts it and it can be easily connected by tapping the Larix shortcut button if on a mobile phone, or by scanning a QR code from an external device.

Another focus was responsiveness which is achieved by connecting a web socket so that the backend sends Server-Side-Events whenever a change happens to any of the streams. Any client device which has the website running with Javascript enabled will be listening to “/SLTUpdate” as a “text/event-stream” and Event emitters are setup up on the backend to send the list of updated streams in JSON form whenever stream CRUD updates happen or the status changes.

5 EVALUATION

5.1 Evaluation method

The results of the project are evaluated through manual testing and meetings with the project owner. For making sure the product is functional there is a procedure intended to serve as a checklist. It outlines the actions administrators need to perform as preparation for the test and the actions and assertions that the tester will perform in order to verify that the application is working as intended. This procedure is detailed in section 3.5.

In addition to the manual test procedure, the plan was to include limited unit test coverage. Some features can be difficult to unit test and due to the strict development time constraints caused by the long research process, this has not been a priority. Unit testing has seen some very limited use in this project, but functions that require mocking and other more complicated test methods are only tested through the manual test procedure.

Finally, and perhaps most importantly, the product is evaluated based on feedback from the project owner. It would be ideal to perform real user testing, but this is difficult to organize and does not seem important for a prototype. The primary concerns in this case are that the application does what it is supposed to do, and how easy it makes it for the user. The former is easy to test, the latter would benefit from user testing.

However, it is also possible to quantify in terms of how many actions the user needs to perform. This has been chosen as a measure of how successful the project is – the fewer steps it takes to start streaming, the more successful the project is. This will be referred to as the “ease of use” evaluation.

5.2 Evaluation results

This section will go over the various ways in which results are acquired, as well as the collected results and their implications.

5.2.1 Manual testing

Manual testing has been performed primarily by the student team, but the application has also been tested at larger scales with the project owner. The primary evaluation method between iterations of the project, as explained in Section 3.5, is a formalized test procedure. The intention of this procedure is that it will serve as a series of actions and assertions a tester should make, in order to verify that all existing functionality is working. As new functionality is added, the procedure is updated in order to make sure it continues to cover all functionality.

This has not been utilized in large-scale testing due to lack of time, however it has been used to verify the functionality of the project whenever a new version was ready.

5.2.2 Unit testing

Unit tests were planned but have not been implemented. This is primarily due to the difficulty of designing good tests for such a complicated system, and time constraints. It would be possible to implement unit test coverage of most functions, but there has not been enough time to do the necessary research and planning on top of the extra work.

5.2.3 Feedback and “ease of use” evaluation

The project has been developed in close cooperation with the Project Owner. Most features and plans have been discussed during meetings before implementation, then demonstrated and discussed after they are implemented. This ensures that the Project Owner has a good overview of how the project is developing and what it is becoming, allowing them to interject early and often in case anything does not align with their vision. This, in turn, ensures that the product being built aligns with the initial goal of the Project Owner.

In addition to ensuring that the product being built is the correct product, a quantitative measure of “ease of use” has been suggested by the project owner. This is a number, representing how many steps a user needs to perform to start a stream, after going to the website. The goal for this metric is 2 – One click to start the stream on the website, and one tap on your phone to start the actual streaming from Larix. It does not appear to be possible to automate the step of starting the stream in Larix. This could be solved by using another application for streaming – possibly developing a custom solution.

At present, the application requires that a user writes a name before they can start a stream. This means we currently have a “ease of use” metric of 3, however it is trivial to implement a name generator or other naming system which can eliminate this one step. This has not been implemented, because there are many solutions and all of them are simple. For example, if a login- or authentication system was implemented, the default stream name could simply be the user’s name. For these reasons, the goal of minimizing actions is considered achieved, even though it has not been directly implemented.

6 RESULTS

6.1 Solved problems

The result of the project is a prototype of an application which satisfies the goal of enabling quick and easy video streaming from mobile devices into a closed NDI system. The application automates almost every step involved in creating a new video streaming pipeline from a user's perspective. A user only needs to visit the website, enter a name for their stream, submit it and then start streaming video by pressing a button or scanning a QR code. This satisfies the primary goal of the project.

6.2 Unsolved problems

The most significant unsolved problem for this project is that the transcoder appears to be struggling with dropped frames. The way this issue manifests when the application is in use is that the video stream (as viewed on the receiving end) stops for a few seconds, and then starts again. This can happen as frequently as multiple times per minute. The issue has been explored and found to be caused by Viz One Coder. The problem has been submitted as a bug report and has been forwarded to them team assigned to Coder.

Beyond the issues with the transcoder, all major problems are solved, and all features built into the application are working correctly as detailed in Section 5.

7 DISCUSSION

7.1 Approaches and consequences

This section will discuss the different approaches to various problems as explained in Section 4, and the consequences those choices have had on the result.

7.1.1 Streaming protocol

It was decided early on to use SRT since it was determined to handle bad streaming conditions better than any other protocol but as outlined in 6.1 this was not working during the development. This meant we had difficulties determining which bugs were coming from the backend and which came from the transcoder. The application is also quite limited in customizing a transmission as this was not sufficiently testable.

The chosen protocol is less documented and offered less flexibility in the use of existing platforms than an older specification would have. Additionally, it locked the client application away from being run in a browser. Choosing a different protocol may have directed more time at developing a browser streaming client and the backend may have been built with a more team-familiar environment.

7.1.2 Development language and environment

React was chosen as the framework for the front-end as it is a very powerful and convenient way to quickly develop responsive web applications. In order to avoid problems associated with working with different languages, Node.js was chosen as a platform for the backend, and express.js was used to develop the backend API.

Given these tools, there are two primary candidates for a programming language: JavaScript and TypeScript. TypeScript was chosen for this project due to its excellent type system, which while providing some challenges in having to learn it, also solves many problems related to code quality and maintainability.

7.1.3 Stream forwarding

The application uses a small C++ application named srt-live-transmit.cpp to forward streams. Being able to receive an SRT data stream and forwarding it to another location is a key feature in the application. This feature was the subject of most research done in the early phases of the project and solving it marked the beginning of the development phase. Many alternatives were explored and are explained in Section 3.

7.1.4 Transcoding

For transcoding, the data stream to the NDI format, the project owner has provided their own proprietary software, known as Viz One Coder – often referred to as “Coder”. Coder exposes a REST (Representational State Transfer) API which can be used to control it in various ways, such as setting up, starting, stopping, monitoring and editing transcoding jobs. The application communicates with Coder through the API and automatically creates and manages jobs as necessary – the user does not even need to know that Coder exists. One consequence of this choice is that, as discussed in Section 6, Coder’s support for SRT

is a fairly new feature and as such it still has some problems. The good news is that it is still in active development, and we can talk to the developers about these issues.

7.1.5 Design

The overarching design of the application is also a matter of importance. One example of how the design has changed in order to satisfy the requirements of the project is that the original plan was to have the front-end communicate with Coder. This would not have worked because Coder will be behind a firewall on a private network, whereas the front-end runs in the user's browser. As such, Coder is not accessible from the internet and has to be accessed from the back-end which serves as a bridge between the public internet and a hypothetical customer's private network. This has caused us to move as much functionality as possible to the back-end, which in retrospect would have been the better decision either way.

8 CONCLUSIONS AND FURTHER WORK

8.1 Goals

The goal of this project as outlined in the project description given by the project owner is:

Exploring the possibilities of streaming video directly from a web application on a smart device, into Viz Story in a simple and user-friendly way so that the content can be processed and published as quickly as possible.

8.2 Further Work

For the solution to be production-ready further work must go into user management, login, preferred profiles, stream security etc. Stream security could be easily configured using SRT encryption capabilities and was simply not introduced as it did not make sense without the other security measures in place.

Installation of the solution is not as streamlined as it would need to be for a full release. Currently the repository requires a pre-compiled platform specific SRT-live-transmit component to be available and path setup trough the config files. It is also necessary to install Coder separately. Some work could be done to make these steps more automatic and not require the user's input.

Long term it is possible to use the Larix Broadcaster SDK (Softvelum, 2021) to provide a customized client for Android and IOS. As this SDK is provided as a one-time purchase the project owner has shown some interest to not rely on any third-party applications. Creating a custom client could also allow for better use of the simultaneous recording feature provided within the SDK. The live stream could be backed up while recording so that a full quality upload could be made later when stable internet connection is available.

Finally, the SRTLiveTransmit tool used is not recommended for production-ready applications and some work could go into using the NPM package explored or making a custom extension to the SRT protocol. At the moment there doesn't seem to be a huge benefit to this, but it may be required for some more advanced SRT setups.

Feedback from the project owner indicates that they do not have any concrete plans for the project to continue, they are comparing this solution to other projects worked on internally and need to assess the marked situation and need for the tool. The project might lay still for a year or more. They communicated an overall satisfaction with the project and the concrete alternative it provided.

9 References

Cinegy GmbH, 2021. *Nuget.* [Online]
Available at: https://www.nuget.org/packages/SrtSharp/1.4.1.207-beta
[Accessed 03 06 2021].

Eyevinn, 2021. *npmjs.* [Online]
Available at: https://www.npmjs.com/package/@eyevinn/srt
[Accessed 03 06 2021].

Fiedler, G., 2017. *https://gafferongames.com/.* [Online]
Available at:
https://gafferongames.com/post/why_cant_i_send_udp_packets_from_a_browser/
[Accessed 03 06 2021].

Google, 2021. *https://webrtc.org/.* [Online]
Available at: https://webrtc.org/
[Accessed 27 05 2021].

Haivision, 2021. *GitHub.* [Online]
Available at: https://github.com/Haivision/srt

Haivision, 2021. *https://www.haivision.com/.* [Online]
Available at: https://www.haivision.com/resources/white-paper/srt-protocol-technical-overview/
[Accessed 03 06 2021].

Haivision, 2021. *SRT-Live-Transmit.* [Online]
Available at: https://github.com/Haivision/srt/blob/master/docs/apps/srt-live-transmit.md
[Accessed 27 05 2021].

Material-UI, 2021. *Material-UI.* [Online]
Available at: https://material-ui.com/
[Accessed 03 06 2021].

NewTek, 2021. *https://www.ndi.tv/about-ndi/.* [Online]
Available at: https://www.ndi.tv/about-ndi/
[Accessed 27 05 2021].

NGINX, 2021. *nginx.com.* [Online]
Available at: https://www.nginx.com/resources/glossary/nginx/
[Accessed 03 06 2021].

NGINX, 2021. *NGINX.com.* [Online]
Available at: https://www.nginx.com/resources/wiki/extending/api/
[Accessed 03 06 2021].

NGINX, 2021. *RTMP-NGINX.* [Online]
Available at: https://www.nginx.com/products/nginx/modules/rtmp-media-streaming/
[Accessed 27 05 2021].

S. O'Dea, 2021. *Mobile operating systems' market share worldwide from January 2012 to January 2021.* [Online]

Available at: https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/
[Accessed 03 06 2021].

Softvelum, 2021. *Larix Broadcaster SDK*. [Online]
Available at: https://softvelum.com/larix/
[Accessed 02 06 2021].

Softvelum, 2021. *Nimble Streamer API.* [Online]
Available at: https://wmspanel.com/nimble/api
[Accessed 03 06 2021].

Softvelum, 2021. *Softvelum.* [Online]
Available at: https://softvelum.com/larix/grove/
[Accessed 03 06 2021].

Softvelum, 2021. *Softvelum.* [Online]
Available at: https://wmspanel.com/nimble
[Accessed 03 06 2021].

Softvelum, 2021. *WMSPanel API.* [Online]
Available at: https://wmspanel.com/api_info
[Accessed 03 06 2021].

Vizrt, 2021. *https://www.vizrtgroup.com/.* [Online]
Available at: https://www.vizrtgroup.com/about/
[Accessed 03 06 2021].

10 APPENDIX

10.1Risk list

See risk management.

10.2 GANTT diagram

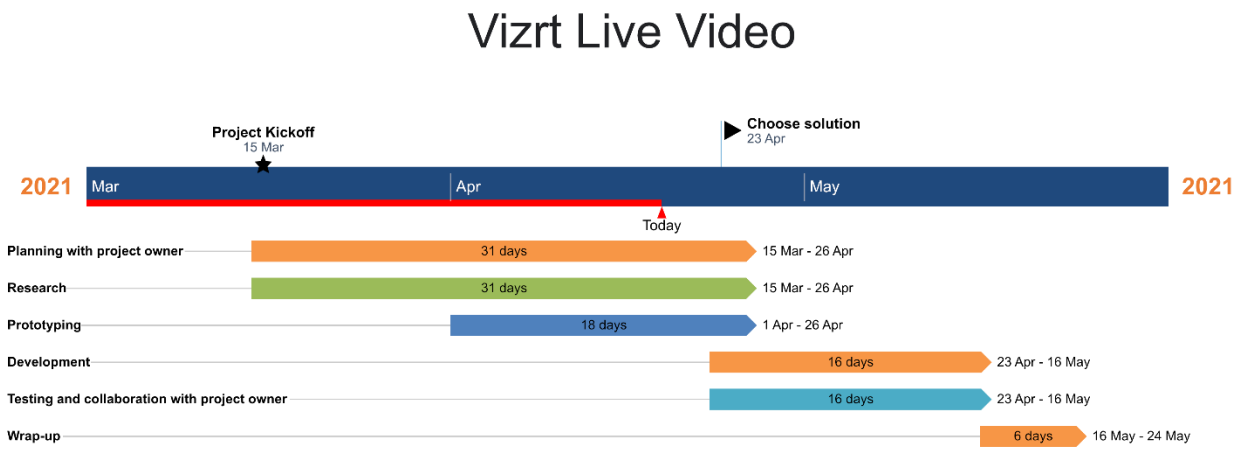

Figure 2: Gantt Diagram

10.3 User manual

10.3.1 Setup

```
git clone https://github.com/vizstory/livevideo
cd livevideo
npm ci
cd frontend
npm ci
```

Clone repository and use npm ci to install dependencies.

10.3.2 Usage

Command	Parameter	Description
npm run	backend	Runs the Express server
	frontend	Runs the React frontend
	dev	Runs both
	test	Runs tests

10.3.3 Configuration

Use Config files located at “backend\config.json” and “frontend\src\config.json” to configure connection between frontend,backend and coder.

CoderProfiles, LarixProfiles and Srt Connection parameters can be configured in “frontend\src\utils\profiles.ts”

(Additional profiles require setup through Coder UI)

10.3.4 Using Frontend

StreamStartPanel

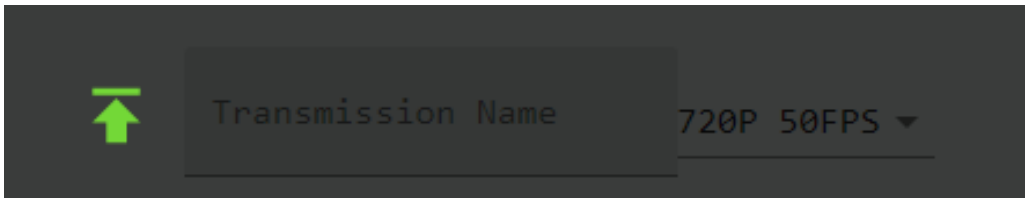

Figure 5: Frontend StreamStartPanel

Start a stream by pressing green arrow, default profile and startbehaviour are configurable in frontend config.

StreamRow

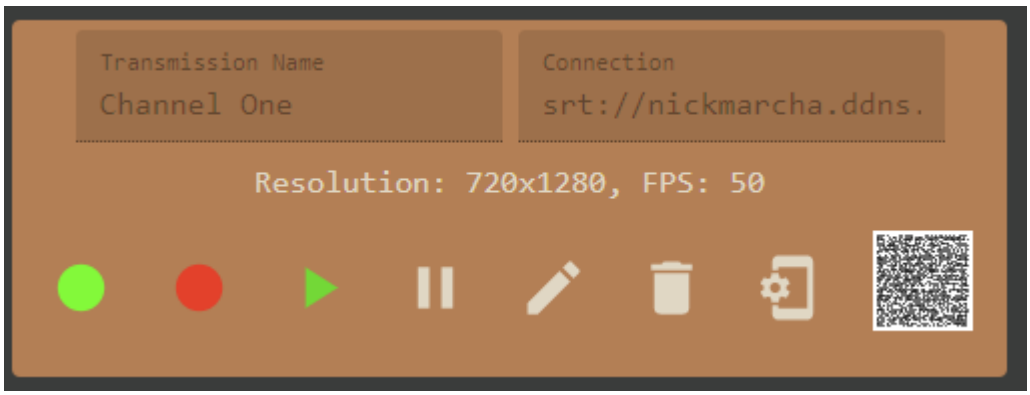

Figure 6: Frontend StreamRow

- Click to copy Name (NDI output) or Connection (SRT uri with params, mode is missing -> caller:default)
- First indicator: whether stream is started or not
- Second indicator: backend SLT connection to source and target
- Start
- Stop
- Edit (currently only NDI output name)
- Delete
- Open Larix shortcut for smartphones (needs Larix Broadcaster installed)
- QRCode: Click to enlarge or shrink (needs Larix Broadcaster installed)

(Mouseover or longtap on mobile to see button explanation)

10.4 Class Diagrams

10.4.1 Frontend

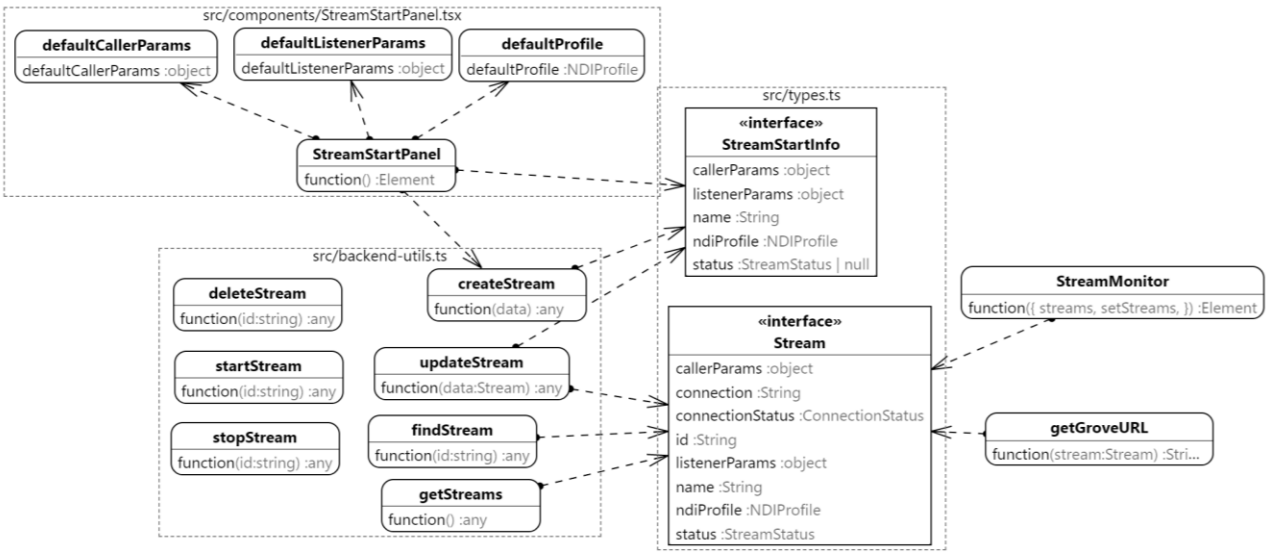

Figure 7: Class Diagram Frontend

10.4.2 Backend

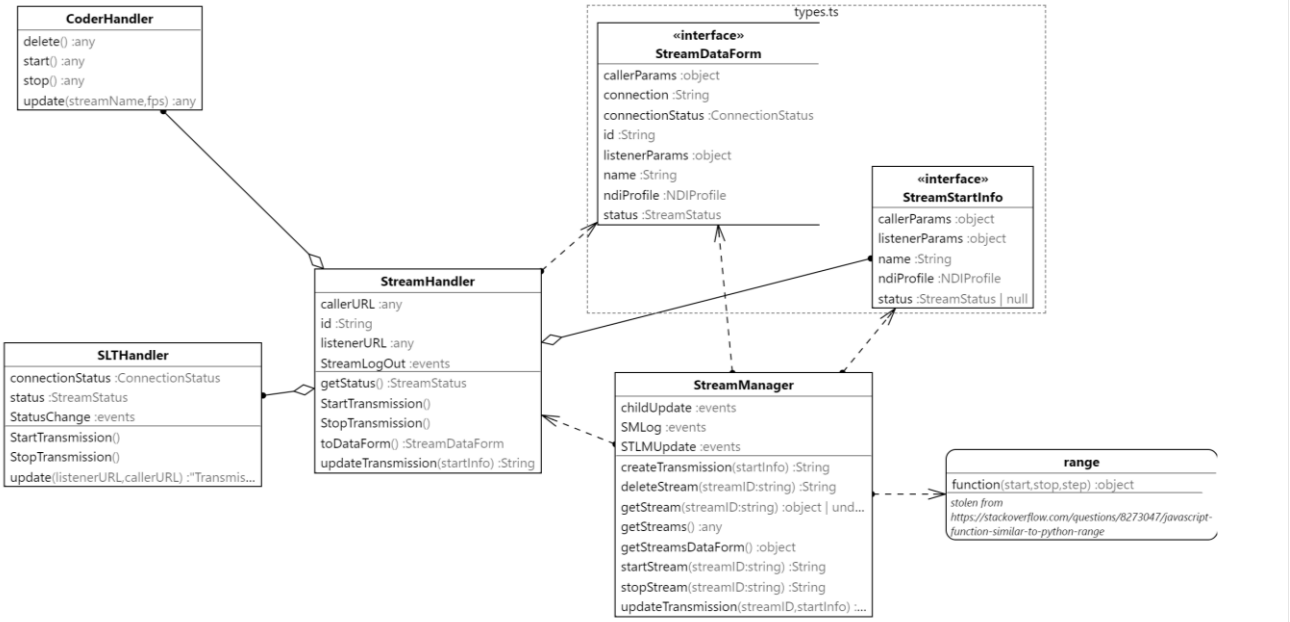

Figure 8: Class Diagram Backend