Western Norway
University of
Applied Sciences

# BACHELOR'S ASSIGNMENT

## HTML graphics for video editor

**Brage Sekse Aarset**
**Jørgen Ullebø Hjartøy**
**Marensius Bae Pettersen**

Computing
Faculty of Engineering and Science
Atle Birger Geitung
04.06.2021

# Tittelside for hovedprosjekt

| *Rapportens tittel:* HTML-grafikk for videoredigeringsverktøy | *Dato:* June 3, 2021 |
|---|---|
| *Forfatter(e):* Brage Sekse Aarset, Jørgen Ullebø Hjartøy, Marensius Bae Pettersen | *Antall sider u/ vedlegg:* 31<br>*Antall sider m/ vedlegg:* 31 |
| *Studieretning:* Dataingeniør | *Antall disketter/CD-er:* 0 |
| *Kontaktperson ved studieretning:* Atle Birger Geitung | *Gradering:* Ingen |
| *Merknader:* | |

| *Oppdragsgiver:* Vizrt, ved Mikal Henriksen | *Oppdragsgivers referanse:* |
|---|---|
| *Oppdragsgivers kontaktperson:* Knut Arvidsson | *Telefon:* |

*Sammendrag:*
Rapporten omhandler prosessen ved å lage et program som fasiliterer bruken av HTML-grafikk i Viz Story. Rapporten tar for seg forberedelsene, selve prosessen og evalueringen av utført arbeid.
*Summary:*
This report concerns the process of creating a program that facilitates the use of HTML graphics in Viz Story. It explores the preparations, the process of development and the evaluation of finished work.

*Stikkord:*

| | | |
|---|---|---|
| Grafikk | Media | Video |
| Webapplikasjon | Viz Story | Vizrt |
| Vektorgrafikk | Agile | Scrumban |

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

## Preface

The source code for the project will not be publicly available, as it is property of Vizrt. Besides, testing the application requires access to some internal test environments.

We would like to thank the stack overflow community for providing boundless knowledge and troubleshooting assistance.

We would also like to extend our thanks to the people advising us on the development of the program and the writing of this report:

- Atle Birger Geitung
- Knut Arvidsson and the Vizrt Story team

Finally we would like to thank the Node.js open source community, who carry the modern web on their shoulders.

# Contents

# 1 Introduction

When producing video content media companies nearly always use some sort of graphical overlays. They can be as simple as a logo in the corner of the screen or more complex graphics with animations. These graphics are used in both live broadcasting and prerecorded videos. Most media companies use third party software for the production of these graphics.

Vizrt is a company based in Bergen that provides such software. They offer a suite of software that facilitates the creation of sophisticated graphic templates that allow the user to quickly edit a video, populate the templates with data and publish to different platforms and formats through an automated process. In order to shorten render times for the graphics, this software uses GPU hardware acceleration and runs on powerful servers. These templates, used in the video editing suite Viz Story, are created in the asset creation tool Viz Artist and configured for production in Viz Story Admin.

## 1.1 Motivation and goal

The goal of this project was to create a less hardware demanding alternative to the existing software for creating graphics templates. A solution utilizing HTML graphics would mean less powerful and cheaper servers could be utilized, which in turn makes the software more scalable.

## 1.2 Context

The current solution for creating graphics is to use powerful and expensive dedicated render farms. This is necessary for live broadcasting where every second counts, and is also necessary for 3D graphics where a lot of processing power is needed to render the graphics. However, when editing prerecorded videos this can be an unnecessarily powerful and expensive approach.

Vizrt's push towards HTML based graphics has produced software components that allow for the usage of HTML graphics in Viz Story. However, the only method for creating the HTML graphics templates is through programming in JavaScript and CSS. This is a time-consuming and difficult process for designers, who usually work with graphical tools like Adobe Photoshop. Vizrt therefore desired a GUI based workflow that could create and edit the HTML graphics assets. The software had to work with the existing software from Vizrt and be able to communicate with components in Viz Story and Viz Story Admin.

## 1.3 Limitations

The greatest limiting factor for this project was the time frame. The project was limited to a timeframe of three months, where alongside the development of the application a lengthy report was to be written and numerous other tasks were to be completed.

It was difficult to predict how much the team would be able to do during the time frame, and therefore the team decided to get a basic functioning demo working and add additional features after that, if time allowed. The visuals of the user interface, as in making it appear visually like a Vizrt program, was not a priority.

Handling secure user logins, secure storage, insertion attack countermeasures and other such issues would not take priority in the project. The focus was on making a functional prototype with a core feature set that could be refactored and expanded by Vizrt's development team later on.

The team did not intend to test the application with its intended end users, which are Vizrt's customers.

## 1.4 Resources

Vizrt was an important resource to the team. Their team of developers working on Viz Story were made available to the team and they showed great willingness to aid in any way they could.

The team was guided by Atle Briger Geitung on issues related to the composition of this report and other formal tasks related to this project.

Due to the pandemic lockdown, the team worked from home and communicated internally through Microsoft Teams and Canvas. Communication with Vizrt was done through Microsoft Teams, and with Atle through Canvas and Zoom. If the situation around the pandemic had changed, Vizrt provided the option of working from their offices in Media City Bergen.

As there were no other practical options, the team decided to utilize their own personal computers for the programming, communication and documentation that the project entailed. The team decided to use Visual Studio Code as the development environment, due to its familiarity to the team. Vizrt provided a GitHub repository with some code from previous projects and various documentation. The team decided to actively use GitHub for source code management as well as work coordination, through a project board connected to the repository. In order to make future integration and further development as easy as possible the team chose to utilize Vue, which is the frontend web framework used by Vizrt in existing products.

## 1.5 Organization of the report

**Chapter one** introduces the project, it describes the goals of the project as well as its limitations. The chapter gives insight to the motivation for the project and gives an introduction to the company behind the project.

**Chapter two** goes more in-depth about the project, it states the initial requirements of the project as well as the initial thoughts about a solution. Furthermore, the chapter enlightens the reader as to existing solutions, previous works and the used reference literature. The project owner and their

role in the project is also be described in the chapter.

**Chapter three** explores the different approaches for solving the project goal. Without going into great detail it confirms the chosen path, and describes the necessary tools. The planed process for development is explained in the chapter, along with an assessment of possible risks. Lastly the manor in which the project is to be evaluated is described.

**Chapter four** describes the overall architecture of the application, and the communication pipeline. It then dives deeper and provides a rundown of the implementation of the program.

**Chapter five** evaluates the project and its result, it contains the internal evaluation and the feedback from the project owner.

**Chapter six** discusses the processes of the project. It looks through the decisions made and their impact on the project. It expresses what was done well and what could have been done better.

**Chapter seven** is the conclusion of the project, whether or not the team achieved the goals of the project. Other uses for the program and the experiences had through the project is discussed, along with the teams suggestions for future work.

**Chapter eight** is the literature list.

# 2 Project Description

This bachelor project is comprised of the development of an application at the behest of Vizrt. The application aimed to ease the production of HTML-graphics for use in Viz Story.

## 2.1 Practical background

The project team and Vizrt had not previously collaborated, but the team had experience with JavaScript and various web application principles. The initial project briefing was quite open, but through meetings and discussions the team and Vizrt found a more concrete goal.

### 2.1.1 Project owner

The company behind the project is Vizrt. For more than 20 years they have supplied digital graphics to the media industry. They employ more than 700 people, with 30 offices worldwide. (Vizrt, 2021a)

This project was organised by the Story team at Vizrt. The project owner of this branch was Mikal Henriksen. The main contact for this project was Knut Arvidsson, who is the team lead of the Story team. Others involved with the project were Arne Bjørsvik Kråkenes, principal developer of Story, Even Normann, senior UX designer, and Roger Rebbestad Sætereng, R&D manager.

### 2.1.2 Previous work

In the development of any product it is important to gain knowledge of previous works or similar solutions that can improve the development or the result. This project was built as an addition to an existing prototype solution for creating and using HTML graphics in Viz Story.

The existing process involved programming graphics by hand, which is not ideal for the end user, namely designers in media organizations. Designers likely prefer working in a graphical user interface that allows for a responsive and visual process. This project would use parts of the existing solution, such as a JavaScript API called "Payload Hosting" that is served alongside an HTML file to allow Viz Story to use the template in a dynamic fashion (updating text, colours, etc. in real time). Viz Story, including Viz Story Admin was made available to the team through a locally runnable test version of the program, which would be used for testing.

Vizrt provided the team with both examples of hand written animated HTML-graphics and SVG example files created in Photoshop. The example files were useful as guides for how the output of the program ought to look and what to expect as input, respectively.

#### Existing solutions

Vizrt has a large software ecosystem with multiple components, and naturally there was no perfect existing solution that would slot perfectly into the existing architecture. However, some similar

software that offered a lot of the required functionality was found. During online research, the team came upon an application called Ease Live Studio. It is an editor where you can create HTML based graphics to use as overlays for video. The team viewed this editor as a possible starting point for the project. Unfortunately the company behind Ease Live was recently acquired by one of Vizrt's competitors, and it was thus deemed to be undesirable avenue for the project to pursue (Ease Live, 2020).

Several solutions were discovered that could perform parts of the desired functionality. There were many available editors that could create assets to use as graphics in HTML. Bitmap images could naturally be included in HTML files, but are less flexible than resolution independent and easy to manipulate vector graphics. The SVG format (Scalable Vector Graphics) is supported in modern web browsers and can be manipulated as a DOM (Document Object Model), and is a natural choice for HTML based dynamic graphics (Mozilla Developer Network, 2021). Among the most popular tools for creating such graphics is Inkscape, which is an open source vector graphics editor. Inkscape is usable on Windows, Mac OS and Linux and uses SVG as its main format (Inkscape Website Developers, 2021).

Adobe Photoshop is a popular editing software that allows exporting of images to SVG format. Adobe Illustrator is a natively vector-based graphics software, similar to Inkscape.

When it comes to animating graphics in HTML/JavaScript/CSS, which is a desired functionality, solutions exist such as SVGator, Keyshape and Haiku Animator. Which are existing editor that lets the end user add animation to SVG's.

### 2.1.3 Initial requirements specification

The initial requirements were to find and integrate or create an application that could make HTML-graphics compatible with Viz Story.

The team was given loose reins as to how the solution would look, but with three main demands:

- Find or create an editor that can export simple graphics to an HTML format.
- Improve the process of graphic design compared to the current hand-programmed process.
- Integrate with template system used in Viz Story

### 2.1.4 Initial Solution idea

The initial solution idea was to make a fully fledged editor that allowed the user to create and edit HTML-graphics in a template format, to then export them to Viz Story.

# 3 Project design

Throughout this chapter the different development approaches will be laid out and the one which offered the highest chance of success will be highlighted. The different tools and software needed for the selected approach will be introduced, and the work plan for the project will be shown as well as the risk assessment.

## 3.1 Possible approaches

Vizrt provided the team with details around the currently extant manual process that this project was to replace, but no specifications or requirements regarding the implementation itself was provided. This made it possible for the team to approach implementation from different angles using a modern technology stack.

### 3.1.1 Alternative approach 1: Build an editor

The first approach the team envisioned was to create a fully fledged standalone editor to complement Viz Story where designers could create vector assets then add text and motion, before exporting it as a template to Viz story.

### 3.1.2 Alternative approach 2: Find and implement an editor

Another considered approach was to find an existing editor with the desired features as described in the first approach, and integrate it into the template generating process in Viz Story.

### 3.1.3 Alternative approach 3: Import assets from third party editors

The final potential approach was to create a simple editor where designers could import assets created in vector graphic editors of their choice as an SVG file. This editor would allow limited editing and integrate into the Viz Story pipeline leveraging the Payload hosting API.

### 3.1.4 Discussion of alternative approaches

The first approach was formulated as the team delved into the issue and tried to come up with the best possible solution. An HTML-based equivalent to Viz Artist would intuitively achieve all of the goals. Viz Artist has been developed and maintained by Vizrt for years and its feature set reflects the requirements of their customers. Certain features could be excluded for the initial version of the editor, and some features would probably be cut altogether (such as 3D modelling).

The biggest perks of this approach would be to have an environment where Vizrt could provide a streamlined user experience and the most effortless way to create templates for designers. The biggest drawback of this approach is that it is very large in scope. Creating a graphics editing software suite is a vast undertaking that usually takes years and requires a lot of experience (X,

2019). Given the limited time frame the team concluded that it would be unrealistic to attempt to create an alternative to Viz Artist in this fashion.

Therefore, the team started to look at integrating an existing vector graphics editor, as described in the second alternate approach. The team had a hard time finding an existing editor that had all the desired core features and that seemed realistic to integrate into Viz Story. Due to most solutions not being open-source software, modifying an existing editor to fit the needs of this project would be hard to achieve. Licensing software for modification is potentially an expensive approach, and not feasible for a no-budget bachelor project.

Given the time constraints, the team concluded that the best approach was to reduce the scope of the project and create a simplified editor, where the user could import SVG files created in their graphics editor of choice (Inkscape, Photoshop, Illustrator, etc.). The user would get the most intuitive and efficient workflow by creating assets in a familiar environment. The software created through this project, HTMLGFX, would act as middleware and prepare the graphics for use as a template in Viz Story. In HTMLGFX the designers could oversee some template logic such as overseeing auto-detected dynamic fields (text fields, colour values, etc).

Additional features to add to the template, such as animation, would be a part of a proprietary editor in HTMLGFX. All the resulting metadata would be included in the exported template, which would be seamlessly exported into Viz Story and used through the Payload Hosting API. The API requires that the graphics are accompanied by a script defining a set of methods for how the graphics are to be dynamically manipulated during video editing and rendering. This is called a VDF model, and should be automatically generated so that the designer does not have to write code.

This approach seemed to be viable based on tinkering with some SVG files created by designers at Vizrt using Adobe Photoshop. The structure of an SVG file varies slightly from editor to editor, but any valid SVG file will be parsable as an SVG DOM. Some interpretation logic would be required to make sense of the hierarchy of elements in the SVG, as different editors were not consistent in the hierarchical organisation of the graphical elements of the exported SVG.

## 3.2 Specification

HTMLGFX Feature list

- Import SVG
- Parse SVG into an Object retaining all tags and data
- GUI components that can modify SVG properties
- Support for different height and width (Aspect Ratios)

- Auto generate VDF models based on templates
- Support for multi format template
- Support for layouts (Drag and drop for moving SVG elements)
- Add motion to SVG elements
- Add GUI for grouping and organizing SVG elements
- Support for Google fonts for a consistent end user experience
- Support for custom fonts
- Integration into Viz Story

## 3.3 Selection of tools and programming languages

### 3.3.1 Tools

#### Visual Studio Code

Visual Studio code is a freeware integrated development environment (IDE) from Microsoft, with support for multiple programming languages and was chosen due to an extensive library of plugins to make it easier to develop software. E.g. GitHub integration and linters for TypeScript & Vue (Microsoft, 2021c).

#### GitHub

GitHub is a service based on Git that hosts & versions the codebase in the cloud. As Vizrt provided the development team with their sample in GitHub, and GitHub has some nice extra features as Actions and Project Board, the development team decided to keep using GitHub (GitHub Inc., 2021).

#### Microsoft Teams

Microsoft Teams was the preferred platform of communication for Vizrt, and as such the development team decided to use Teams internally as well (Microsoft, 2021b).

### 3.3.2 Programming Languages

#### TypeScript

As the development team were familiar with working with strongly typed languages such as Java, Typescript was chosen over JavaScript. Which is basically a superset of JavaScript with a strongly typed syntax (Microsoft, 2021a).

### 3.3.3 Frameworks

#### Vue

Vue is a frontend framework which provides the developer a scaffold and a toolbox to build a web application. Vue was chosen with consistency and support in mind, as Vizrt was using this framework in their existing products (Vue, 2021).

#### Express.js

A minimal and flexible Node.js web application framework was chosen to serve static files due to it's easy setup, configuration and TypeScript support (Express, 2019).

#### Storybook

Storybook is a tool that can be implemented with Vue to create an environment where developers can create isolated frontend components (Storybook, 2021). The team opted to try out the frontend development experience of Storybook.

### 3.3.4 Additional software

#### Viz Story

A web-based video editing suite created by Vizrt. It uses templated 3D animated graphics and allows the user to quickly edit and publish video content to multiple platforms (Vizrt, 2021c).

#### SVGson

An open source library to parse SVG into JSON and serialize it back to SVG (elrumordelaluz, 2021).

## 3.4 Project development method

The team consists of three developers. Given the small size of the team and that the product needs continuous feedback from the project owner, an agile approach was chosen in order to have a flexible goal and the ability to readjust the process according to new information and developments along the way (Beck et al., 2001).

The development of the HTMLGFX was broken down into incremental stages or iterations with each iteration bringing a new set of features. Implementation of these stages was done in stages or sprints where larger features were broken down to sprint sized tasks.

Progress was tracked using a Kanban board (GitHub Project Board). This allowed the team to visually prioritize features, assign tasks and keep track of the sprint progress.

Sprint reviews with Vizrt were held in order to get feedback, improve continuously and prioritize the backlog.

Scrum was held for developers in order to update each other and delegate tasks, and discuss any obstacles that might arise.

The development method approach can be summarized as Scrumban, leveraging the best tools from Agile, Scrum and Kanban. (ProjectPlan, 2021)

### 3.4.1 Initial project plan

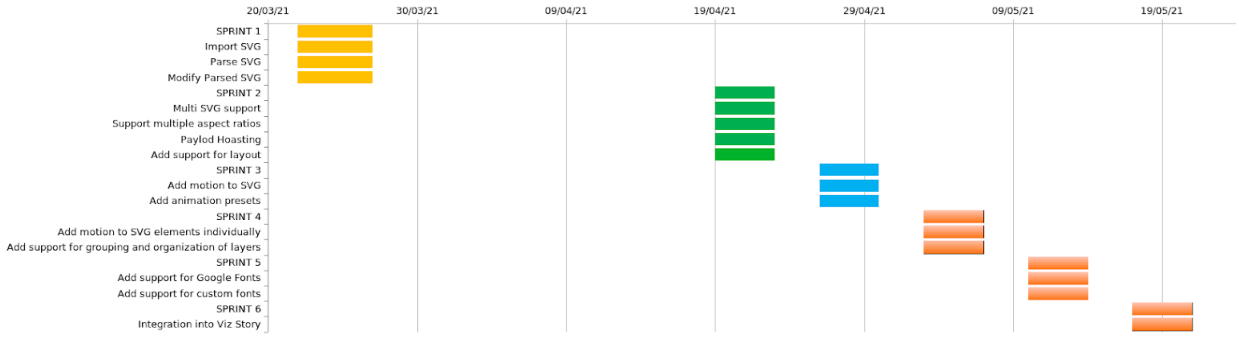


**Figure 3.4.1-1:** The original GANTT chart.

As seen in the initial project plan (figure 3.4.1-1), various iterations were planned before development started, each containing a new set of desired features.

### 3.4.2 Risk management

The most significant risk was deemed to be that the project scope could be too big. The team had little basis for determining a realistic estimate of its own output over time, which could lead to the scope being larger than the time frame and resources would allow. This could have lead to an inability to deliver the core features.

Development was done in a transparent way through GitHub and regular meetings with the project owner and stakeholders, thus exposing risks early and handling them as they arise. By having the sprint reviews with Vizrt the knowledge of all parties involved was harnessed, enabling detection of more risks that the development team internally could not find. Another perk of involving the project owner through the whole development process was to agree on customer needs and reduce the risk of delivering the wrong product. Additionally risks were exposed internally in the development team during daily scrum.

## 3.5 Evaluation method

The HTML Graphics software is designed to be used by graphic designers and video production staff, but it was unlikely that the product would reach a state within the project period where user testing would be appropriate. Therefore it was more realistic that Vizrt's Viz Story team would do the testing and evaluation of the program. Since Vizrt hopefully would use the code as a basis for a mature product, it was important that the code was readable and extensible. Therefore a technical evaluation from Vizrt's developers would be central to the evaluation.

In this project the linter ESLint is used, which gives direct feedback about different code quality issues, such as TypeScript typing and other issues. This instant feedback was used throughout the project to maintain code quality, but would also be used to inform the final evaluation.

Additionally, in this report, the team will give its on evaluation of the results, as well as provide an extended discussion about the experiences made during the project period. This report could be a useful resource for Vizrt, but an evaluation of the report itself cannot exist within the report, as this would be a circular paradox.

# 4 Detailed design

## 4.1 The architecture

### 4.1.1 Processing graphics

The main purpose of the application with the chosen approach, is to allow a user to import SVG graphics, specify which parts of the graphics are dynamic, and export it to Viz Story. A pipeline architecture lends itself to this approach. In a pipeline architecture, data is sent through a linear sequence of steps, each performing their own actions on the data in chronological order (Data Pipelines, 2021). There is an input (an SVG file) that is to be processed in various ways through a linear process where the resulting output must conform to the specifications of Viz Story. Thus, there will be tight coupling between each step of the pipeline. Steps along the pipeline may have non-linear actions the user can take, but the output of those steps must nonetheless match the specifications of the succeeding part of the pipeline.

The pipeline is visualised in figure 4.1.1-1. The "File import and parse" process is explained in section 4.2.1. The rendering of the SVG is merely a visualisation for the convenience of the user and doesn't do anything with the data therefore not a part of the data pipeline itself. See section 4.2.2. This graphic is not intended to show which components perform which tasks. This is merely a visualisation of the data processing pipeline. In `htmlgfx-editor` there is an editor component that is the parent of all other components and owns the state of the application. The component hierarchy is considered unnecessary to reach a sufficient understanding of the application design and is not outlined in this report.

The "User input" process consists of the user selecting which elements of the graphics are supposed to be dynamic. See section 4.2.3 and section 4.2.4. The "Payload preparation" process consists of multiple steps and is explained in section 4.2.5, section 4.2.6 and section 4.2.7. The communication with the API is detailed in section 4.1.2. On the API side, the HTML page building process is explained in section 4.2.8.

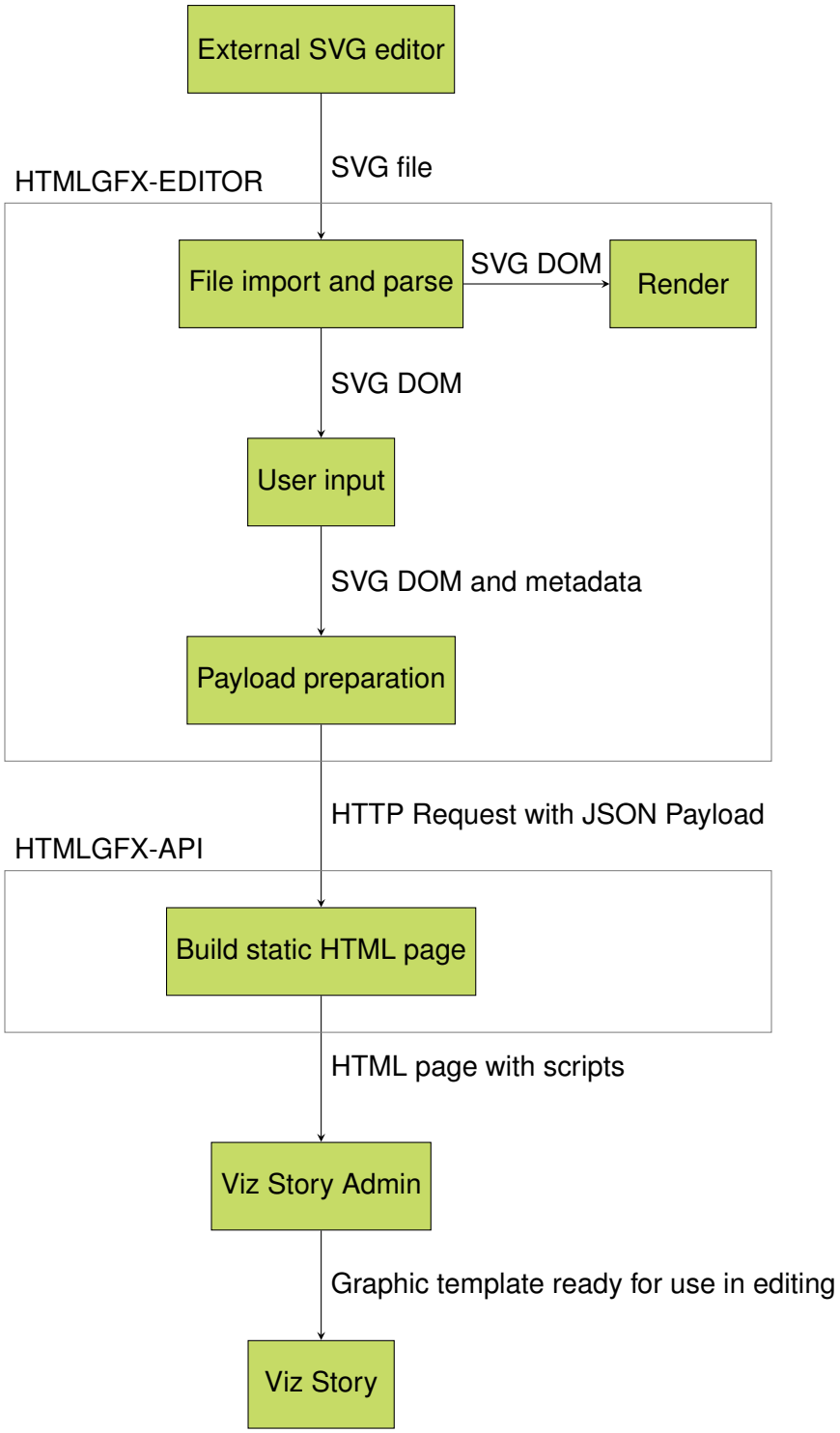


**Figure 4.1.1-1:** The data pipeline of the project.

### 4.1.2 Component communication

`htmlgfx-editor` and `htmlgfx-api` share some of the codebase in order to reduce development time and to keep things simple. The library vdfModelMethods.js is used both in the editor and the api to manipulate the SVG.

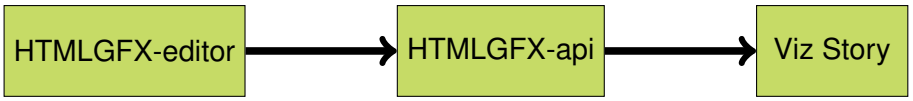


**Figure 4.1.2-1:** Component Communication

The communication consists of three components, `htmlgfx-editor`, `htmlgfx-api` and Viz Story.

The `htmlgfx-editor` communicates with the `htmlgfx-api` using the HTTP standard, with a simple POST request containing the SVG + metadata. The `htmlgfx-api` takes the POST request from `htmlgfx-editor` and creates a static website that can be served to Viz Story for consumption, based on the SVG and metadata it receives from the `htmlgfx-editor`.

`htmlgfx-api` provides Viz Story the payload by serving static files, which the user enters with a URL. It uses the VDF model (see section 4.2.5) in the metadata to call callback methods (see section 4.2.6) in the static website, thus allowing the end user to manipulate the graphics in real time.

## 4.2 Implementation

### 4.2.1 Importing the SVG

An SVG file, unlike raster images, is simply a text document (XML-based structure) describing the vector image. SVG can be read and written just as though it were HTML markup, and, also like HTML, has a DOM. It is therefore a well suited format for working with vector graphics in web applications, and is widely supported by web browsers, languages and frameworks (Mozilla Developer Network, 2021).

In order to import an SVG file, the team opted for the simplest method that would involve the least additional dependencies, namely to use HTML's built-in `<input>` element with the type set to "file". This prompts the user to upload a file from their file system, which is then handled by JavaScript through an event handler using the built-in file API to read the text content of the uploaded file. After the text content is imported, the DOMParser interface parses the text into a DOM Document so that the SVG can be manipulated in various ways.

### 4.2.2 Rendering the SVG

In order to show the user a preview of the SVG before exporting to Viz Story Admin, the team had to look at what Vue has to offer when it comes to rendering DOM content from an imported file. Vue *can* render raw HTML content using the `v-html` directive but this is considered a bad practice due to the potential for cross-site scripting attacks (Lee, 2017).

The alternate approach is using Vue's `render()` function. Normally when creating a Vue component, the visuals are written in a HTML-like template syntax that is styled with CSS and made dynamic through data binding. For this specific purpose, however, the DOM itself needs to be

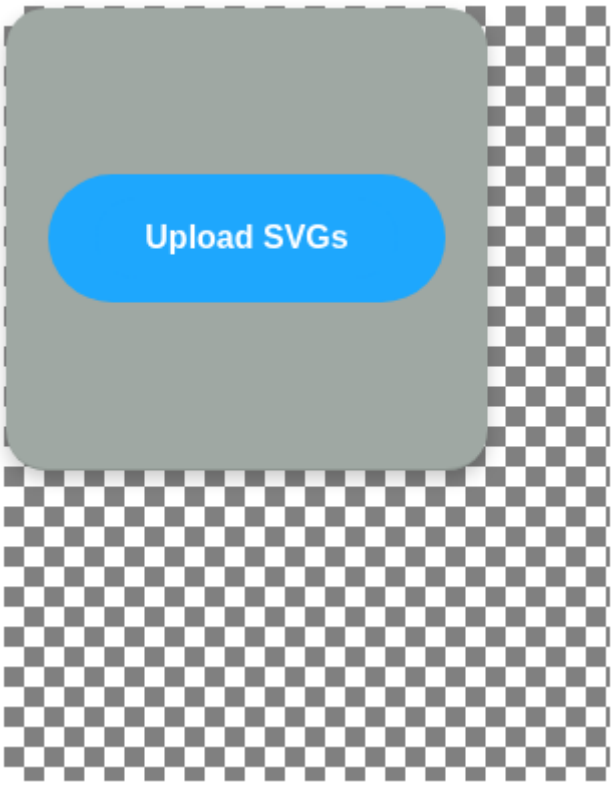


**Figure 4.2.1-1:** The upload button in the editor.

constructed from the data, and the render function is the way to achieve that.

Without going into too much detail, the SVG render component takes in an SVG DOM, traverses it recursively and renders every element and its children to the Vue DOM. Attributes are carried over (id, style attributes, etc.), and so is innerHTML content. The result is a perfect recreation of the original SVG file. Due to how Vue works with object props passed to components, the component will not automatically update when the SVG DOM is updated. As it is desirable to have a "live" view of the SVG that updates as changes come in, the parent component that owns the SVG DOM forces the render component to re-render whenever changes are made, using a key attribute.

### 4.2.3 Selecting dynamic elements

After import, the user has the option to select elements of the graphics the user intends to be dynamic. A dynamic element is a part of the graphics where attributes of their choice such as fill colour, text content, position etc. can be changed in the video editing suite. A typical example in the broadcasting industry is a "lower third", which is a graphical element in the lower part of the screen displaying information such as the name of a person on screen (StudioBinder, 2020).

The selection of which elements to mark as dynamic consists of selecting the element from a list and then selecting which attributes of that element are to be dynamic.

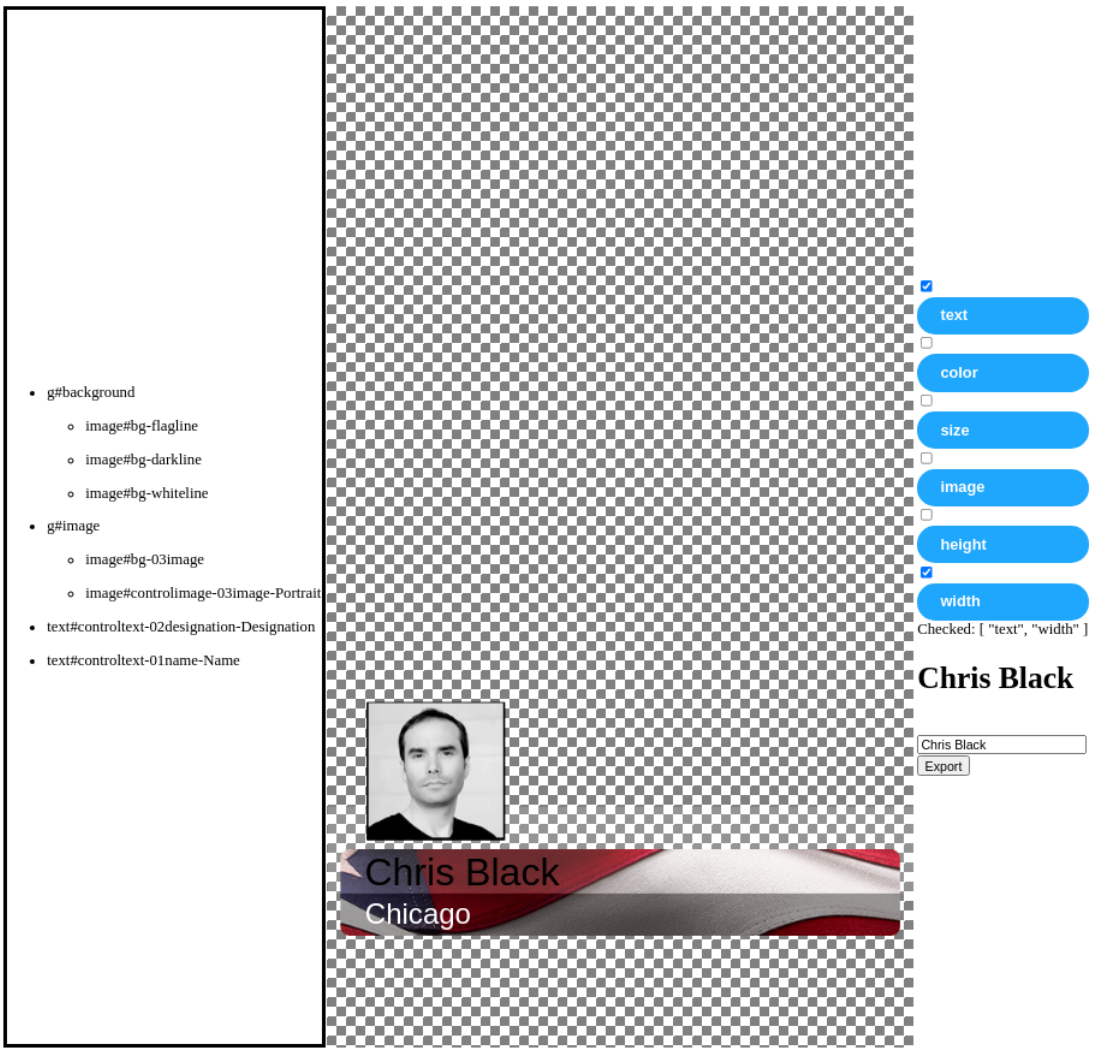


**Figure 4.2.2-1:** The editor with an imported SVG graphic rendered on the screen. The list of elements in the SVG is shown on the left side, and the attributes of the currently selected element on the right side.

This list is created by a recursive component that traverses the hierarchy of the SVG DOM and creates a hierarchical list of tagnames with their ids. This list can be see in figure 4.2.2-1. When clicking an element in this list, an event is transmitted to its parent component, and the event is propagated throughout the hierarchy in the list until it reaches the main editor component. This is an example of a perhaps needlessly complex event propagation system that could warrant the usage of a state management solution in the future.

### 4.2.4 Selecting dynamic attributes

The selection of dynamic attributes is dealt with by the component VdfCheckboxes. Once the user has selected an element from the list described in section 4.2.3, it becomes visible and active. The VdfCheckboxes component contains a checkbox for each attribute, and a list of the selected attributes. Using the checkboxes the user select the attributes they would like to be dynamic. Currently, the type of element or the existing attributes of the selected element is not taken into account.

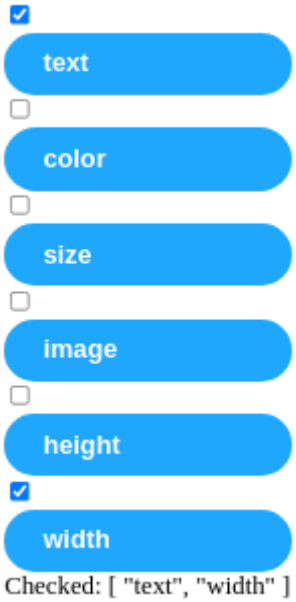


**Figure 4.2.4-1:** VdfCheckboxes component in the editor

The list of selected attributes is bound to a variable in the main program. When the user checks a checkbox, the VdfCheckboxes component emits the change to the main program, and the list storing the selected attributes is updated.

The currently selected SVG element and the list of chosen attributes is stored in a list in the main program. This list is only updated when the user hits the export button, or clicks on an element in the element list described in section 4.2.3.

As the list containing selected elements and attributes is stored in the main program, a way of communicating with the component is a necessity. The parent component owns the state of the

program, and the child component must retrieve the state information as it is needed to relay that information to the user. This is achieved through using the Vue option **watch** to update in response to changes of the state.

### 4.2.5 Generating VDF-model

Viz Story Admin needs something called a VDF-model to be able to interact with the HTML graphics. Vizrt states:

> "The Vizrt Data Format is a specification used internally at Vizrt that standardizes how our data is shaped. For our purpose of making HTML Graphics it is used primarily for sending data back and forth between a host application and the graphic for the purposes of playing the animations and / or setting values." (Vizrt, 2021b)

Generating the VDF-model uses specialized methods for each editable attribute. The current method for selecting elements and attributes is described in section 4.2.3 and 4.2.4. The user repeats this process until all desired elements and attributes has been selected.

```
import * as methodModel from '../vdfModelMethods';

const vdfModel = document.implementation.createDocument(null, "model");

/**
    Generates a vdf model XML element for a text field.
    @param tag - the element having its vdf generated
*/
export function newTextField(tag: SVGSVGElement): HTMLElement {
  const field = vdfModel.createElement("fielddef");
  const valueElem = vdfModel.createElement("value");
  valueElem.textContent = tag.innerHTML;
  field.setAttribute("name", `${tag.id}-${methodModel.updateText.name}`);
  field.setAttribute("label", tag.id);
  field.setAttribute("mediatype", "text/plain");
  field.setAttribute("xsdtype", "string");
  field.appendChild(valueElem);
  return field;
}
```

**Figure 4.2.5-1:** Code example for generating VDF

Once the list, containing elements and attributes described in section 4.2.4, is submitted, a method in the program generates the VDF as an XML element. The XML can be viewed as having two parts, the user selected parts and the base parts. The user selected parts are child elements of the base parts, therefore the method generating the VDF-model first generates the base parts before using a for-loop and a switch case to call the specialized methods for each editable attribute.

The team discussed three different ways of generating the VDF-model, making a string, using

HTML and using XML. Although the method in the payload hosting API that sets up the VDF-model takes a string this approach was quickly discarded as it was the least structured alternative. The difference in structure between using HTML and XML is minimal, but their uses differ. XML is used for describing data and HTML is used for displaying data (GeeksforGeeks, 2020). This, alongside the fact that the payload hosting API sets up an XML model, made the team choose generating XML.

### 4.2.6 General methods for editing the image.

The payload hosting API uses callback methods to change the HTML graphic when the user makes changes to the graphic in Viz Story, such as updating text content. In the examples supplied by Vizrt the methods were manually written for each graphic. These methods were used as a guide to make the general methods.

The general methods need the id of the element (referred to as `object` in the source code) being changed, it then finds the element in the document and updates its attribute with the provided value.

```
/**
 * Updates the color for the object with the given id
 * @param newColor - The color to be used
 * @param objectId - The id of the object that is being changed
 */
export function updateColor(newColor, objectId) {
  if (!newColor) return;
  document.getElementById(objectId).style.fill = newColor;
}
methods[updateColor.name] = updateColor;
```

**Figure 4.2.6-1:** Method for updating color

The callback methods used by the payload hosting API only allows for the passing of one variable, this is an issue when using the general methods. In order to solve this the team implemented anonymous functions that take the parameter from the callback and add its id to the function call of the correct general method.

The team only managed to implement functioning methods for modifying text and color.

The general methods are used on the API to set up the scripts required by Viz Story. In order to communicate to the API which methods are to be used for a given VDF model field, the methods are mapped to unique string identifiers that are passed in the payload, and mapped back to methods on the API by indexing the string identifiers in an object.

### 4.2.7 Building the payload

`htmlgfx-editor` creates a payload to be processed in `htmlgfx-api` based on user inputs.

```
interface JsonPayload {
    list: FieldValues[];
    vdfModel: string;
    svg: INode;
}
```

**Figure 4.2.7-1:** Structure of JsonPayload

The payload contains three elements:

1. **list:** Contains information about which SVG elements can be manipulated, and which callback method they map to.
2. **vdfModel:** A base64 encoded representation of the VDF Model.
3. **svg:** A json representation of SVG.

### 4.2.8 Building static HTML

`htmlgfx-api` has a library that generates a HTML document based on the payload received in the `htmlgfx-editor` POST request. It uses the third party library jsdom to create a HTML document with the SVG from the editor embedded. `htmlgfx-api` also generates the necessary JavaScript files using the metadata from `htmlgfx-editor`.

Viz Story Admin supports importing HTML graphics when provided with a link to a static HTML page that contains certain JavaScript code. The payload hosting API is one of the JavaScript files served with the HTML page.

In figure 4.2.8-1, the Viz Story Admin template editor is shown, with a lower third imported from an SVG file through HTMLGFX. Two text elements are dynamic; the text content can be changed to reflect the name and location of the person in question.

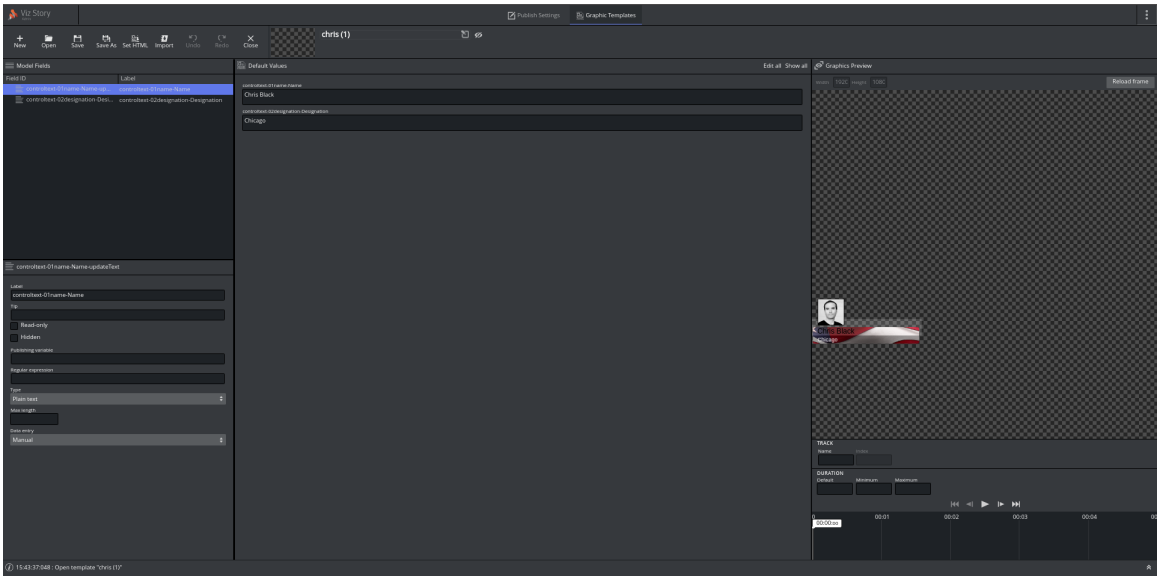


**Figure 4.2.8-1:** HTML graphics loaded to Viz Story Admin

# 5 Evaluations

## 5.1 Evaluation method

The group will evaluate the work by assessing how many goals were achieved, the quality of the code, the quality of documentation and general readability and re-usability of the code.

Vizrt's Viz Story team will evaluate the project by looking at the code and by using the program itself to test its capabilities.

They will assess the success level of the product through metrics similar to those the team will use to evaluate the project. Essentially, the product ought to be something Vizrt can use as a proof of concept and a starting point for further development. The viability of this hinges on the code being understandable and that the project takes a clear and conscious approach to solving the goals of the program. It may be the case that the chosen approach is laden with bugs or is not sufficiently extensible such that major rewrites of the code will be necessary. In that case, the project would not necessarily be a failure if there are valuable takeaways that will improve the second version. The Wright brothers made hundreds of failed prototype flying machines before their first successful powered flight, after all (Library of Congress, 2021).

## 5.2 Evaluation results

### 5.2.1 Internal evaluation

The result of the work is a working prototype in accordance with the feature set of the first iteration. The most important technical goal has thereby been reached. The goals of each subsequent iteration were not met. There are a few particular issues with the application that ought to be highlighted.

#### Naming

There are several components and other files that could have benefitted from some refactoring. Naming conventions were not always central to the development process, and this shows in some cases. There is particularly one component that has a misleading name due to intended functionality that was scrapped to save time.

#### Duplicated code

There is a JavaScript file that is used by both the `htmlgfx-editor` and the `htmlgfx-api`, and currently this file is duplicated in each repository so that any changes made have to be manually performed twice. This is obviously a bad practice, and there are many different ways to solve this.

#### JavaScript vs. TypeScript

The aforementioned JavaScript file has no business being written in JavaScript when the related code is mostly comprised of TypeScript. Converting the file to TypeScript would have made typing more consistent and would have resolved some complaints from the linter.

#### Linter warnings

The latest linter run resulted in 22 warnings, mostly relating to unexpected uses of the any type. The cases where any has been used were deemed unavoidable, which typically arise from frameworks not providing types, such as in the case of Vue's render method. Same warnings are due to known ESLint bugs.

#### Documentation

Almost all methods, interface, types and classes in the code have inline documentation written for them in doc comments. Some less involved methods lack doc comments as it can be unnecessary in cases such as getters and setters. The team is satisfied with the documentation.

#### Project plan compliance

Throughout the project, it became obvious that the project scope needed a re-evaluation. Most tasks took longer than excepted. The project scope was quickly readjusted and the focus became solely on the core functionality.

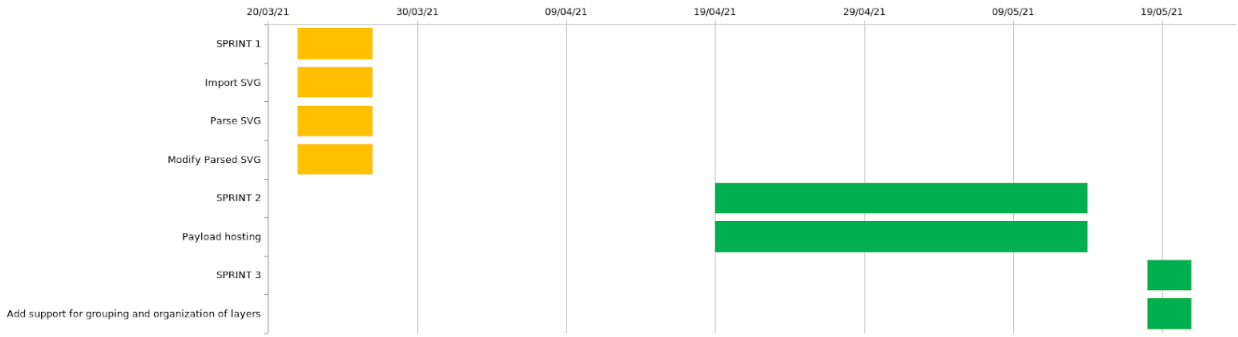


**Figure 5.2.1-1:** The revised GANTT chart.

The revised plan almost came to fruition (organising and grouping of layers was never implemented). See figure 5.2.1-1.

### 5.2.2 Evaluation from Vizrt

After the end of the development period (mid May), the program was demonstrated to Vizrt. The team also presented some information about the architecture and the development process.

Vizrt's feedback was overall positive. They were satisfied with seeing a working proof of concept which will inform their future research and development in this area. They did not expect more than a basic prototype within such a short time frame.

The source code itself was restructured, cleaned up, documented and finalized the same week, but the Vizrt team did not have time to provide feedback on the source code itself within the report deadline.

# 6 Discussion

## 6.1 Chosen frameworks

### 6.1.1 Storybook

The initial idea with Storybook was to have a systematic approach to creating decoupled Vue components using the Atomic design methodology.

Atomic design methodology is the idea of breaking down a "view" / "screen" into a system of components.

- Pages - E.g. Webpage with content
- Templates - E.g. Structure / Layout of Webpage
- Organisms - E.g. Header
- Molecules - E.g. Search section of header
- Atom - E.g. Search button

Storybook allows the developer to write "stories" which is in short different states, in order to isolate components, and opens up for working on one component at a time. This was the main motivation behind using Storybook. Allowing the team to work on different components simultaneously.

The team believes that initial intentions were good, but due to not prioritizing the visual design of the editor and the fact that the team was creating a system that was so coupled that few components could be tested in Storybook on their own, the team did not leverage Storybook as intended. It was a good learning experience, but in hindsight the team added overhead without reaping the benefits in this case.

Although the team felt that Storybook was ultimately not needed for the project at this stage, Vizrt had considered using Storybook in their projects and took interest in seeing a demonstration of it in use with Vue. By coincidence, the project ended up being useful to Vizrt as an example of Storybook being used alongside Vue.

### 6.1.2 Vue

The team was mostly inexperienced with frontend frameworks and the concept of lifecycles and state management. This led to a slow start in development and the team experienced a steep learning curve.

One of the main challenges for the team was handling communication between components, and dealing with propagation of events throughout the component hierarchy to send state information

to relevant components. A state management framework such as Vuex would probably have been useful, and will probably be more and more relevant the bigger the application grows in the future.

Throughout the project the team experienced that using a frontend framework was the correct choice, as it made it easier to work systematically and focus on creating the editor. Building everything from scratch would have been tedious and time consuming.

## 6.2 What could have been done differently

### 6.2.1 Time management

As this was the first time the team worked together and on a project of this magnitude, a big risk was not having historical throughput to base time estimates on. This led to the scope of the project growing too big and a significant readjustment of the scope taking place mid-project. This could be improved next time by identifying the core features more precisely and focusing solely on those.

Being inexperienced developers working in separate locations and with two of the three team members working a job on the side, sometimes a steady rhythm in the work was not easy to maintain. Scheduling conflicts posed a challenge to our daily meetings, which in turn induced a loss of productivity. The experimental nature of the project also made it difficult to assess the time needed to complete specific tasks.

Furthermore, the amount of tasks not related to the actual development of the product proved to be greater and more time consuming than first assumed. Being more efficient in completing and more aware of these tasks would at the very least mean a better estimate of available time for the project.

### 6.2.2 Communication

Vizrt was accommodating since day one, yet the team hesitated to ask technical questions to Vizrt when it could be avoided by doing research on the web, which was often time consuming. In the final meeting with Vizrt they expressed that this is a common issue with junior developers, where they would rather "bang their heads against the wall" for a while than ask for help. The team believes that by leveraging Vizrt's expertise more, the development process could have been accelerated.

Regrettably, the team was unable to work physically at Vizrt's offices due to the pandemic. If the team had worked at Vizrt's offices, some of the issues in communication may have been lessened. The threshold for asking for help may have been lower and Vizrt may have initiated more communication, such as asking how it's going during lunch.

This could have been solved by having more frequent communication in Teams, and by having weekly sync meetings as planned. Due to poor time management and schedules that did not line

up, these meetings were dropped.

### 6.2.3 Research and decision making

The team's development philosophy was quite exploratory, and decisions were made based on if this decision would present an issue or limitations in the future as the scope of the project grew or not. Some limitations posed by the choices made in this period may become apparent years down the line, and some might have become obvious had the team continued the work for a few more weeks. All in all, the project has quite few external dependencies and the scope of the project makes it quite feasible to remove dependencies such as Storybook.

#### Greater exploration of possibilities at each step

The team worked in a Scrumban fashion which led the team to handle abnormalities as they came up. But in hindsight the team should have implemented the concept of "Jidoka" more, which consists of:

1. Discover an abnormality
2. Stop the process
3. Fix the immediate problem
4. Investigate and solve the root cause

(Kanbanize, 2021)

The team should have had more focus on the last step, and reconsider each time if the team was moving in the right direction.

# 7 Conclusions and further work

## 7.1 Conclusions

To summarize the team's achievements, the team was able to develop a proof of concept where the team proved it possible to implement the process Vizrt was researching. With the workflow being as follows:

1. Designer creates a graphical asset in a software of their own choosing.
2. They export the graphical asset as a SVG.
3. Imports the SVG into the HTMLGFX editor, which exports to HTML wrapped SVG with VDF.
4. Imports the processed SVG into Viz Story Admin.

This means that the original goal was achieved, although many of the sub goals were not reached. As mentioned earlier in this report, due to the limited time frame and lack of historical output data the scope grew too big. Thus the team are not surprised the team was not able to implement additional features, outside what was needed to prove that the process was possible.

## 7.2 Other uses

### 7.2.1 The program

The program developed through this project is very proprietary to Vizrt's systems, as the core functionality is to create a Vizrt Data Format (VDF) XML file and callback methods to manipulate the graphics in VizStory. Therefore the team does not believe the result would be of use to others. However, the experiences gained through this project are something anyone could learn from or at least recognise from their own projects.

### 7.2.2 Experiences

This project gave the team much experience in project and time management. The team fell in many of the known pitfalls of development, such as underestimating the time a feature needs to be implemented. Although the team does not believe the program could be useful outside Vizrt, the lessons learned can be of use to others.

## 7.3 Further work

As discussed in section 5.1, the future development of this program does not necessarily need to build on the actual code in this project, but can reuse specific code fragments or utilise some of the same concepts. Vizrt could also use the experiences from this project us make a more educated choice of a a completely different approach.

If this program is to be further developed the team has some thoughts and suggestions as to what could be done.

### 7.3.1 Security and error handling

As stated in the limitations of the project the security issues and error handling surrounding importing an SVG has not been addressed in the project. This is something that would need solving if the program is to be a commercially viable product. Another limitation to the project that would need attention in future works is the user interface. As Vizrt is an established company they have their own design language, which the user interface would need to comply with.

### 7.3.2 Support for more attributes

The team only managed to implement working functions for editing text content and fill color. Any further development would need to expand on this and add functions for all relevant attributes, such as images, positioning and stroke color.

### 7.3.3 Automation

As explained in section 4.2.4, the selection of VDF-tags is currently done manually and without respect to the type of the selected element. Since few types of editable tags were available in this prototype this is not an issue. However, once the program supports many editable attributes it is likely this will become a cumbersome process that should be streamlined and partially automated. Therefore it would be good to find a way to automatically select the editable attributes, with the ability for the user to change the selection if it does not comply with their needs, and perhaps find seldom used attributes in a separate menu.

### 7.3.4 Other major feature sets

Many desirable features such as support for multiple aspect ratios and support for animations need to be added in the future.

# 8 References


Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R. C., Schwaber, K., Sutherland, J., & Thomas, D. (2001). *Manifesto for agile software development*. Retrieved June 2, 2021, from https://agilemanifesto.org/

Data Pipelines. (2021, February 24). *What is a data pipeline?* [Data pipelines | go beyond integration]. Retrieved May 27, 2021, from https://www.datapipelines.com/blog/what-is-a-data-pipeline/

Ease Live. (2020, October 30). *Sixty announces sale of ease live interactive graphics platform to evertz* [Ease live]. Retrieved May 26, 2021, from https://www.easelive.tv/evertz/index.html

elrumordelaluz. (2021, April 1). *Svgson* [Npm]. Retrieved June 2, 2021, from https://www.npmjs.com/package/svgson

Express. (2019, May 25). *Express - node.js web application framework*. Retrieved June 2, 2021, from https://expressjs.com/

GeeksforGeeks. (2020, October 29). *HTML vs XML - GeeksforGeeks*. Retrieved May 26, 2021, from https://www.geeksforgeeks.org/html-vs-xml/

GitHub Inc. (2021). *Build software better, together* [GitHub]. Retrieved June 2, 2021, from https://github.com

Inkscape Website Developers. (2021, May 27). *About | inkscape*. Retrieved May 28, 2021, from https://inkscape.org/about/

Kanbanize. (2021). *What is jidoka?* [Kanban software for agile project management]. Retrieved May 28, 2021, from https://kanbanize.com/continuous-flow/jidoka

Lee, E. (2017, January 7). *Template syntax — vue.js*. Retrieved May 26, 2021, from https://vuejs.org/v2/guide/syntax.html#Raw-HTML

Library of Congress. (2021). *Wilbur and orville wright papers at the library of congress* [Library of congress, washington, d.c. 20540 USA]. Retrieved May 28, 2021, from https://www.loc.gov/collections/wilbur-and-orville-wright-papers/articles-and-essays/the-wilbur-and-orville-wright-timeline-1846-to-1948/1901-to-1910/

Microsoft. (2021a). *Typed JavaScript at any scale.* Retrieved June 2, 2021, from https://www.typescriptlang.org/

Microsoft. (2021b). *Videokonferanser, møter og anrop | Microsoft Teams*. Retrieved June 2, 2021, from https://www.microsoft.com/nb-no/microsoft-teams/group-chat-software

Microsoft. (2021c). *Visual studio code - code editing. redefined*. Retrieved June 2, 2021, from https://code.visualstudio.com/

Mozilla Developer Network. (2021, March 15). *SVG: Scalable vector graphics | MDN*. Retrieved May 25, 2021, from https://developer.mozilla.org/en-US/docs/Web/SVG

ProjectPlan. (2021). *What is scrumban? | definition, overview, and examples*. Retrieved June 2, 2021, from https://www.productplan.com/glossary/scrumban/

Storybook. (2021). *Storybook: UI component explorer for frontend developers*. Retrieved June 2, 2021, from https://storybook.js.org

StudioBinder. (2020, November 15). *What is a lower third? definition and design strategies* [StudioBinder]. Retrieved May 31, 2021, from https://www.studiobinder.com/blog/what-is-a-lower-third/

Vizrt. (2021a). *About vizrt*. Retrieved May 28, 2021, from https://www.vizrt.com/vizrt

Vizrt. (2021b, February 3). *Htmlgfx/concepts.md at main · vizstory/htmlgfx*. Retrieved May 26, 2021, from https://github.com/vizstory/htmlgfx/blob/main/HTMLGraphicsDocumentation/docs/concepts-vocabulary/concepts.md

Vizrt. (2021c). *Improve your social media branding with vizrt*. Retrieved June 2, 2021, from https://www.vizrt.com/products/viz-story

Vue. (2021). *Vue.js*. Retrieved June 2, 2021, from https://vuejs.org/

X, A. (2019, July 2). *Building a web-based motion graphics editor* [Medium]. Retrieved June 2, 2021, from https://medium.com/women-make/building-a-web-based-motion-graphics-editor-bd070f8db795